920

2047

A139111

ADA139111

Battelle
Columbus Laboratories

Report

GUIDANCE AND ACTUATION SYSTEMS
FOR AN ADAPTIVE-SUSPENSION VEHICLE

M. R. Patterson, J. J. Reidy, R. C. Rudolph
BATTELLE
Columbus Division
505 King Avenue
Columbus, Ohio 43201-2693

TE360 1114

FINAL TECHNICAL REPORT

20020726127

on

GUIDANCE AND ACTUATION SYSTEMS FOR AN ADAPTIVE-SUSPENSION VEHICLE

M. R. Patterson, J. J. Reidy, R. C. Rudolph
BATTELLE
Columbus Division
505 King Avenue
Columbus, Ohio 43201-2693

Sponsored by

Defense Advanced Research Projects Agency (DoD)
DARPA Order No. 4670

Under Contract No. DAAE07-83-C-R040 issued by

U. S. Army Tank-Automotive Command, Warren, Michigan 48090

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER G8186-685-84	2. GOVT ACCESSION NO. AD-A139111	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Guidance and Actuation Systems for an Adaptive-Suspension Vehicle		5. TYPE OF REPORT & PERIOD COVERED 7 March 1983 to 15 March 1984
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Mark R. Patterson, John J. Reidy, and Robert C. Rudolph		8. CONTRACT OR GRANT NUMBER(s) DAAE07-83-C-R040
9. PERFORMING ORGANIZATION NAME AND ADDRESS Battelle Columbus Division 505 King Avenue Columbus, Ohio 43201-2693		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U.S. Army Tank-Automotive Command Warren, MI 48090		12. REPORT DATE March 14, 1984
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Defense Advanced Research Projects Agency DARPA/DSO 1400 Wilson Boulevard Arlingto, Virginia 22209		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Robotic mobility, legged locomotion, rough terrain mobility, vehicle guidance, legged vehicle actuation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This technical report describes the work performed by Battelle Columbus Division on a project which was part of the DARPA Adaptive-Suspension Vehicle program. The Battelle project consisted of two major tasks. The first task's objective was to develop a computer system that would generate vehicle trajectories and leg motion sequences that would enable the adaptive-suspension vehicle to move over rough terrain along a path specified by a human operator. The second task involved the design, fabrication and testing of a safety valve for the foot-lift circuit of the vehicle's leg.

(Continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

TABLE OF CONTENTS (Continued)

SUMMARY

This report describes the work performed by Battelle Columbus Division on a project that was part of the DARPA Adaptive-Suspension Vehicle Program. The Battelle project consisted of two major tasks. The first task involved the guidance of the adaptive-suspension vehicle; the goal was to develop a computer system which would enable the vehicle to traverse rough terrain. The second task concerned the design, fabrication, and testing of a safety valve for the foot lift-circuit of the vehicle's leg. Those two tasks are discussed separately in this report.

Guidance Task

Work on the guidance problem involved the development of a computer system to determine vehicle trajectories and leg motions that would enable the vehicle to traverse rough terrain. The system uses the operator's requested vehicle velocities, information about the vehicle state, and data from a terrain-scanning system in the determination of those trajectories and leg motions.

The hardware on which the algorithms described above are implemented comprises six boards communicating over an Intel Multibus. Three of the boards, one board containing special-purpose circuitry and two Intel iSBC 86/30 processor boards, receive data from the the terrain-scanning system, convert them to elevation information, and then store that information on a 512-kilobyte memory borad. The other two boards are two more Intel iSBC 86/30 processor boards. One of them generates the vehicle deceleration plans using the terrain elevation map stored on the memory board, as well as information concerning the operator's velocity requests and the vehicle state. That information is received from the sixth board, which is responsible for communicaton with the vehicle control computers; it receives the operator requests and vehicle state information and transmits vehicle body and leg motion commands derived from the deceleration plans.

The approach used by the system's algorithms is to maintain at all times a body trajectory and leg motion sequence by which the vehicle can be brought to a halt in a position of static stability. That is, at every point along its path, the vehicle has available a plan for a vehicle trajectory that will bring it to a halt, together with a sequence of leg motions that will allow it to follow that trajectory and will leave it in a state of static stability at the end of the trajectory.

The maintenance of such a trajectory clearly places limits on vehicle velocity over a given terrain. Thus, with this approach, operator requests for vehicle velocities are not always met, although the vehicle will follow those requests as closely as possible while still maintaining vehicle safety. It can be seen that this approach does not permit an inexperienced operator to get the vehicle in trouble, but it still allows an experienced operator to use the full capabilities of the vehicle.

Foot-Lift Safety Valve Task

This task was originally intended to include a revision of the Battelle design of the foot-lift circuit to incorporate experimental data. However, due to a change in the system requirements, a different design was used and the Battelle design was never experimentally evaluated.

The original scope also included a review of the design from a safety standpoint. This task was expanded to the design, fabrication, and testing of safety valve for the foot-lift circuit. This safety valve is intended as a last-resort mechanism for minimizing or eliminating any potential for damage to either the operator or the vehicle itself.

The requirements for the design of the safety valve were generated through discussions with OSU. Based on these requirements, several conceptual approaches were generated and evaluated. After further discussions with OSU personnel, an approach that involved electrically-actuated explosive primer was selected. The valve was then designed and fabricated at Battelle. Tests were then conducted to check the strength and integrity of the design under pressure, as well as its speed of response.

FOREWARD

This research was supported by the Defense Advanced Research Projects Agency, Washington, D.C., under Contract No. DAAE07-83-C-R040. Work was conducted in the period March 1983 through December 1983 by personnel of the Battelle Columbus Laboratories. The principal investigators were Mark R. Patterson of the Digital Systems and Technology Section (guidance task) and John J. Reidy and Robert C. Rudolph of the Equipment Development Section (actuation task).

The project engineers would like to acknowledge the cooperation and assistance of the personnel at Ohio State University involved in the development of the ASV-84 vehicle, in particular Professors Robert McGhee and Kenneth Waldron. In addition, valuable guidance was provided by Dr. Clinton Kelly, the DARPA Project Officer for this program.

1.0 INTRODUCTION

It has long been recognized that most man-made vehicles are greatly inferior to human beings and other terrestrial animals in off-road locomotion. The shortcomings of current vehicles are particularly noticeable in the area of mobility. On rough terrain, a vehicle with a passive suspension system must accomodate obstacles by gross body motions. On the other hand, a system with active suspension units, such as legs, can pick its way through rough terrain by selecting the most suitable footholds and stepping over obstacles and soft spots. In addition, a legged system can compensate for terrain irregularities on which it must step by actively adjusting leg lengths, thus providing a much smoother ride. This report describes work performed in this project as part of a program to develop such a vehicle.

The first part of the report describes the development of the vehicle "guidance" system, which uses information from a terrain-sensing system in the determination of appropriate vehicle trajectories and leg motions. The description of the guidance system has three main sections. The first of those sections describes the vehicle and terrain-sensing system with which the algorithms are intended to be used. The second section presents a description of the algorithms used in the system for conversion of the terrain scanner data to an elevation map and for generation of plans for bringing the vehicle safely to a halt. The last major section of the first portion of the report describes the computer hardware on which the guidance algorithms are implemented.

In the experimental evaluation of the ASV, procedures will be developed to minimize the potential for harm to the operator or damage to the vehicle. As a back-up system, Battelle has designed, fabricated, and tested a safety valve that will be activated in case of a major system breakdown. This effort is described in the second portion of this report.

2.0 GUIDANCE SYSTEM RESEARCH

2.1 Introduction

This portion of this report discusses the development of a computer system that determines the body and leg motions required for a legged adaptive-suspension vehicle to walk over rough terrain. The first section describes the vehicle and its terrain-sensing system. The second section gives a description of the algorithms used in the system for conversion of the terrain scanner data to an elevation map and for generation of plans for vehicle body trajectories and leg motion sequences. The hardware that is used to implement those algorithms is described in the third section. Finally, some conclusions from this research are presented at the end of this portion of the report.

2.2 Background

2.2.1 The Adaptive-Suspension Vehicle

The vehicle for which the guidance system has been developed is approximately 15 feet (4.6 meters) long and 4 feet (1.2 meters) wide. Its height can be varied between approximately 5 feet (1.5 meters) and 9 feet (2.7 meters) by changing the extension of its six three-degree-of-freedom legs. The legs are attached at the top of the vehicle, with one pair each near the front, middle, and rear of the vehicle.

The velocity of the vehicle is expected to be limited, at least on the rough terrain for which the guidance system has been developed, to a translational velocity of no more than 8 feet/second (2.4 meters/second) and a rotational velocity of no more than 30 degrees/second. Translational and rotational accelerations are expected to be limited to no more than 4 feet/second/second/ (1.2 meters/second/second) and 15 degrees/second/second, respectively.

2.2.2 The Terrain-Sensing System

The terrain-sensing system with which the guidance system described in this report is intended to work is a scanning system mounted at the front top of the vehicle. The system scans both in elevation and azimuth, so, for each scan, it provides information for a sector of terrain in front of the vehicle. For each point in its scan, the sending system measures the distance from the scanner to the terrain at its current elevation and azimuth angles.

2.3 Guidance System Algorithms

2.3.1 Overview

As mentioned above, the algorithms for the guidance system perform two distinct tasks. One of those tasks is the conversion of data from the terrain scanner to a terrain elevation map. The other is the generation of vehicle body trajectories and leg motion sequences using the elevation map, the operator's vehicle velocity requests, and information concerning the vehicle's state. The algorithms that perform those two tasks are described in the next two sections.

2.3.2 Elevation Map Algorithms

As described above, the terrain-sensing system provides, for each of its scan points, information on the range to the terrain. Since the scanner is fixed to the vehicle, when the vehicle is moving, each of the scan-point range measurements is made from a different position. Thus, the input from the scanning system to the guidance system consists of scan-point range data indexed by elevation and azimuth angles and measured with respect to the moving vehicle.

However, the form of terrain information most useful for the vehicle guidance algorithms is that of elevations indexed by horizontal positions. For that reason, when a range value is received from the scanner, the elevation map algorithms use the elevation and azimuth angles of the range value, together with the known position and orientation of the vehicle, to calculate the position in earth-fixed Cartesian coordinates of the point indicated by the scanner.

The terrain point positions in Cartesian coordinates are then stored in an array of elevation values divided into cells in a horizontal plane. The algorithms first determine whether there is a cell present in the array for the horizontal position of the terrain point. If so, the elevation value for the point is stored in that cell; if not, a cell is assigned to the horizontal position of the point for storage of the elevation value. Since the terrain array is fixed in size, this method requires that, as the vehicle moves, data storage "wrap around" from one portion of the array to another. Thus, areas of the terrain are automatically "forgotten" after the vehicle has passed some distance beyond them.

2.3.3 Vehicle Guidance Algorithms

The function of the vehicle guidance algorithms is to determine appropriate body and leg motion commands for the vehicle control system, based on current operator requests. Those operator requests can be for three components of vehicle velocity: forward, side (crab), and turning (yaw). The guidance algorithms attempt to match the vehicle velocity to the operator's requests as closely as possible; however, as discussed below, those requests are not always attainable, due to considerations of vehicle stability.

Since the three vehicle velocity components mentioned above are usually the only ones which are of direct interest to the operator, the vehicle guidance algorithms automatically control the vehicle elevation, pitch, and roll based on the terrain over which the vehicle is passing. Then, once all six vehicle velocity components are specified, the guidance algorithms determine the leg motions required to attain those velocities while maintaining vehicle stability. The information required by the guidance algorithms to provide these body and leg motion commands includes, in addition to the operator's requests and the terrain elevation map described in the previous section, information concerning the current vehicle body state (position and velocity) and the positions and support states of the legs, as well as knowledge of the limitations on vehicle velocity and acceleration and on leg motions.

The use of that information in the operation of the guidance algorithms is discussed in the following three parts of this section. The first part presents the general principles on which the algorithms are based and describes the operation of the algorithms' highest level, which determines how closely the vehicle can follow the operator's velocity requests. The second part gives a description of the generation of trajectories for the vehicle body, based on the operator's requests. Finally, the last part describes the generation of sequences of leg motions that allow the vehicle body to follow those trajectories.

2.3.3.1 Vehicle Guidance Algorithm Approach. Previous approaches to the problem of legged vehicle locomotion have emphasized the concept of static stability, the condition in which the vertical projection of the vehicle's center of mass is within the convex polygon formed by the vertical projections of those of the vehicle's feet that are on the ground. Those earlier approaches have used as their goal in vehicle control the maintenance of the vehicle in a state of static stability at all times. This method of control has been possible because vehicle dynamics have been relatively unimportant since vehicle speeds have been quite low.

However, for a vehicle operating at higher speeds, maintenance of static stability is not sufficient for vehicle security, since in many cases the vehicle's motion could carry its center of mass outside its support polygon. Thus, a faster vehicle requires some sort of dynamic control of vehicle stability to ensure its safety. (Of course, for those times when the vehicle is not moving, static stability is still sufficient.)

The approach that is used by the system described in this report is to maintain at all times a plan by which the vehicle could be brought to a halt in a position of static stability. That is, at every point along its path, the vehicle has available a plan for a vehicle trajectory that would bring it to a halt, together with a sequence of leg motions that would allow it to follow that trajectory and would leave it in a state of static stability at the end of the trajectory.

The maintaining of such a trajectory clearly places limits on vehicle velocity over a given terrain. Thus, with this approach, operator requests for vehicle velocities are not always met, although the vehicle follows those requests as closely as possible while still maintaining vehicle safety. The task of determinig how closely the vehicle can follow the operator's velocity requests is performed by the highest level of the guidance system's algorithms.

The algorithms make that determination as shown in the flow chart in Figure 1 (flow chart conventions for this report are given in Appendix A). They first evaluate whether it is possible, for accelerations that would allow the vehicle to attain the operator's current velocity requests as soon as possible, to calculate a vehicle body trajectory and leg motion sequence that would allow the vehicle to be brought to a halt safely. If so, the body and leg motion commands to accelerate the vehicle in the direction of the operator's requests are sent to the vehicle control system. If not, the same attempt (to find a body trajectory and leg motion sequence that would bring the vehicle safely to a halt) is made for each of several successively smaller vehicle accelerations in the direction of the operator's velocity requests. If a body trajectory and leg motion sequence for halting are found for any of those vehicle acceleration (compromise) selections, then the comands to implement that selection are sent to the vehicle control system.

If no part of the operator's velocity requests can be executed safely, then the guidance system derives the commands for the vehicle control system from the most recently generated body trajectory and leg motion sequence. That is, since the operator is requesting that the vehicle accelerate to unsafe velocities (unsafe because the vehicle could not be brought safely to a halt if it were accelerated toward those velocities), the guidance system must ignore the operator's requests and derive its commands from a plan that it knows to be safe, the body

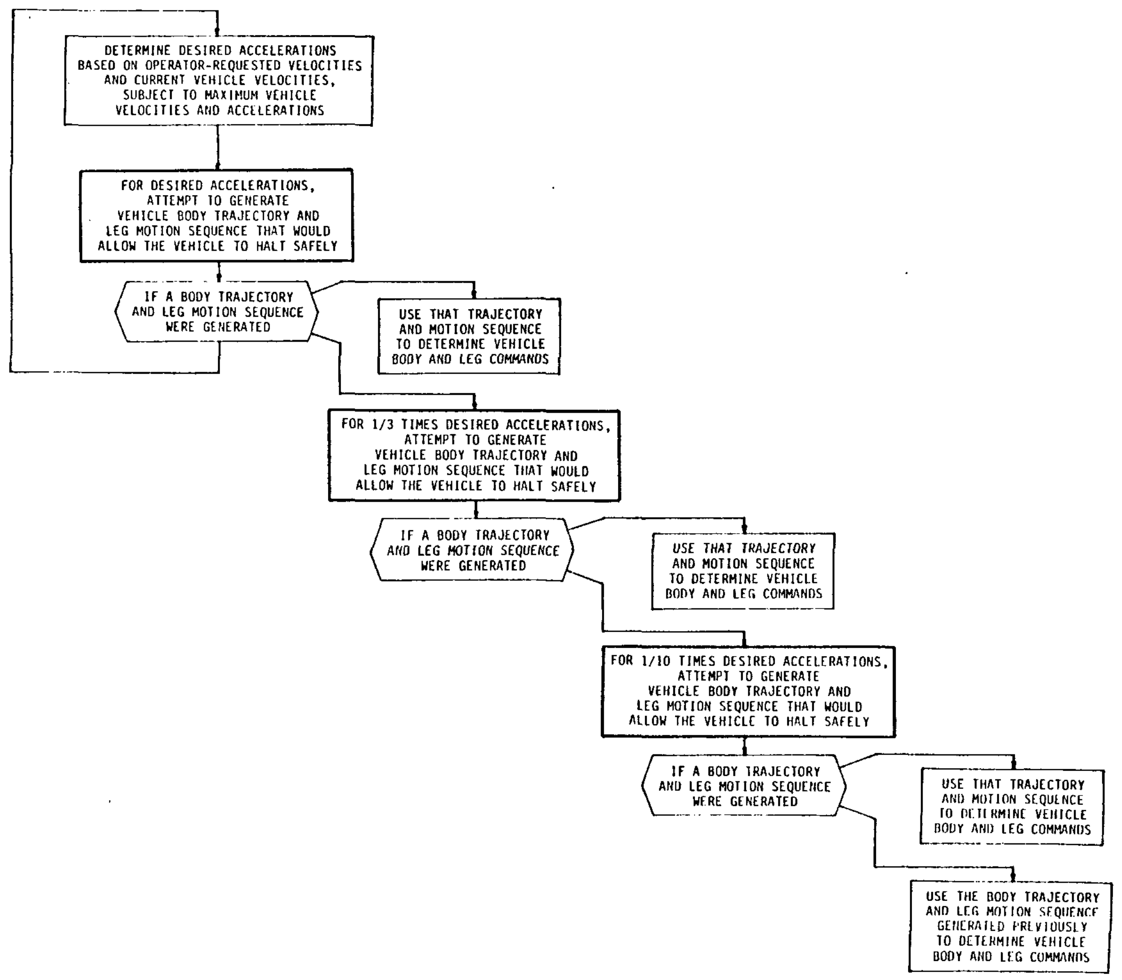

DETERMINE DESIRED ACCELERATIONS BASED ON OPERATOR-REQUESTED VELOCITIES AND CURRENT VEHICLE VELOCITIES, SUBJECT TO MAXIMUM VEHICLE VELOCITIES AND ACCELERATIONS
FOR DESIRED ACCELERATIONS, ATTEMPT TO GENERATE VEHICLE BODY TRAJECTORY AND LEG MOTION SEQUENCE THAT WOULD ALLOW THE VEHICLE TO HALT SAFELY
IF A BODY TRAJECTORY AND LEG MOTION SEQUENCE WERE GENERATED
USE THAT TRAJECTORY AND MOTION SEQUENCE TO DETERMINE VEHICLE BODY AND LEG COMMANDS
FOR 1/3 TIMES DESIRED ACCELERATIONS, ATTEMPT TO GENERATE VEHICLE BODY TRAJECTORY AND LEG MOTION SEQUENCE THAT WOULD ALLOW THE VEHICLE TO HALT SAFELY
IF A BODY TRAJECTORY AND LEG MOTION SEQUENCE WERE GENERATED
USE THAT TRAJECTORY AND MOTION SEQUENCE TO DETERMINE VEHICLE BODY AND LEG COMMANDS
FOR 1/10 TIMES DESIRED ACCELERATIONS, ATTEMPT TO GENERATE VEHICLE BODY TRAJECTORY AND LEG MOTION SEQUENCE THAT WOULD ALLOW THE VEHICLE TO HALT SAFELY
IF A BODY TRAJECTORY AND LEG MOTION SEQUENCE WERE GENERATED
USE THAT TRAJECTORY AND MOTION SEQUENCE TO DETERMINE VEHICLE BODY AND LEG COMMANDS
USE THE BODY TRAJECTORY AND LEG MOTION SEQUENCE GENERATED PREVIOUSLY TO DETERMINE VEHICLE BODY AND LEG COMMANDS

FIGURE 1. VEHICLE GUIDANCE ALGORITHM HIGH-LEVEL FLOW CHART

trajectory and leg motion sequence that it previously determined would bring the vehicle safely to a halt.

Thus, both the safety and the performance of the vehicle are dependent on the guidance system's generation of appropriate body trajectories and leg motion sequences for the vehicle. The next two sections describe the methods which the system uses to generate those body trajectories and leg motion sequences.

2.3.3.2 Vehicle Body Trajectory Generation. As mentioned earlier, the vehicle's operator makes requests for forward, side, and turning velocities for the vehicle. Thus, when the guidance system algorithms attempt to generate a vehicle body trajectory, the vehicle's desired forward, side, and turning accelerations are specified; as described in the last section, they may be accelerations that would move the vehicle most rapidly toward the operator's requested velocities or a smaller acceleration "compromise" selection.

As shown in Figure 2 (again, see Appendix A for flow-chart conventions), the algorithm generates the first portion of the vehicle path over the terrain by assuming that the vehicle will accelerate at the desired rates for the time that passes between successive iterations of the guidance algorithms. The desired acceleration rates determine three of the vehicle's degrees of freedom (horizontal position and yaw angle) at discrete points on the first portion of its trajectory; the acceleration rates are also used to calculate the times at which those points on the trajectory are reached. The algorithm then uses the terrain map to determine the elevation and slope of the terrain at the discrete points along the calculated path. Then the algorithm uses the elevation and slope information together with the desired vehicle altitude and the desired relation of vehicle orientation to the terrain slope to calculate the three remaining degrees of freedom (vehicle elevation, pitch, and roll). Thus, the first portion of the body trajectory consists of vehicle positions (specified by the six body degrees of freedom) at given times for discrete points along the path determined by the desire accelerations.

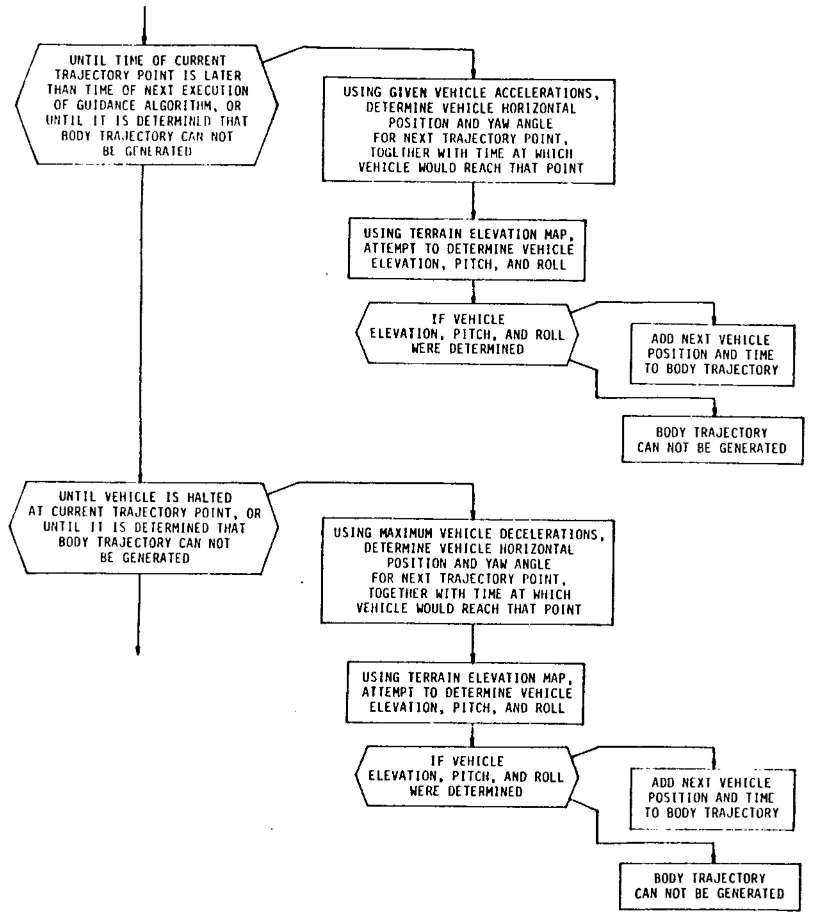

UNTIL TIME OF CURRENT TRAJECTORY POINT IS LATER THAN TIME OF NEXT EXECUTION OF GUIDANCE ALGORITHM, OR UNTIL IT IS DETERMINED THAT BODY TRAJECTORY CAN NOT BE GENERATED
USING GIVEN VEHICLE ACCELERATIONS, DETERMINE VEHICLE HORIZONTAL POSITION AND YAW ANGLE FOR NEXT TRAJECTORY POINT, TOGETHER WITH TIME AT WHICH VEHICLE WOULD REACH THAT POINT
USING TERRAIN ELEVATION MAP, ATTEMPT TO DETERMINE VEHICLE ELEVATION, PITCH, AND ROLL
IF VEHICLE ELEVATION, PITCH, AND ROLL WERE DETERMINED
ADD NEXT VEHICLE POSITION AND TIME TO BODY TRAJECTORY
BODY TRAJECTORY CAN NOT BE GENERATED
UNTIL VEHICLE IS HALTED AT CURRENT TRAJECTORY POINT, OR UNTIL IT IS DETERMINED THAT BODY TRAJECTORY CAN NOT BE GENERATED
USING MAXIMUM VEHICLE DECELERATIONS, DETERMINE VEHICLE HORIZONTAL POSITION AND YAW ANGLE FOR NEXT TRAJECTORY POINT, TOGETHER WITH TIME AT WHICH VEHICLE WOULD REACH THAT POINT
USING TERRAIN ELEVATION MAP, ATTEMPT TO DETERMINE VEHICLE ELEVATION, PITCH, AND ROLL
IF VEHICLE ELEVATION, PITCH, AND ROLL WERE DETERMINED
ADD NEXT VEHICLE POSITION AND TIME TO BODY TRAJECTORY
BODY TRAJECTORY CAN NOT BE GENERATED

FIGURE 2. VEHICLE BODY TRAJECTORY GENERATION FLOW CHART

The second and final portion of the vehicle body trajectory is generated in a similar manner. The vehicle is assumed to decelerate at its maximum rate while continuing on the same path that is was following at the end of the first portion of the trajectory. The deceleration rates are used to determine the vehicle's horizontal position and yaw angle, together with the associated times, for points along the second portion of the trajectory. The elevation map is then used in the calculation of the other three degrees of freedom for those points on the trajectory.

If at any point in the generation of the first or second portion of the body trajectory the algorithm cannot obtain sufficient data from the terrain elevation map to determine the terrain elevation and slope, the attempt at trajectory generation is aborted, which implies that the vehicle cannot implement the desired accelerations. Otherwise, when the trajectory generation is complete, the guidance system proceeds, as described in the next section, to attempt to determine a leg motion sequence which will enable the vehicle to follow the calculated trajectory.

2.3.3.3 Vehicle Leg Motion Sequence Generation. The approach to the generation of vehicle leg motion sequences is shown in Figure 3 (see Appendix A for flow-chart conventions). The algorithm uses an iterative approach that proceeds either until a leg motion sequence is generated for the complete body trajectory (which brings the vehicle to a halt) or until a point is reached at which no acceptable continuation of the leg motion sequence can be found. The iterations of the algorithm take place at the successive discrete points of the previously generated vehicle body trajectory.

For each iteration of the algorithm, the first step is to determine, using the already-generated portion of the leg motion sequence, whether at that point in the body trajectory, if the motion sequence were executed, any of the legs would be completing their transfers from earlier footholds to new ones. If so, it is assumed that those legs would be supporting the vehicle body at that point in its trajectory. The vehicle's static stability is then calculated using the information of the positions of the vehicle's legs that would be supporting its

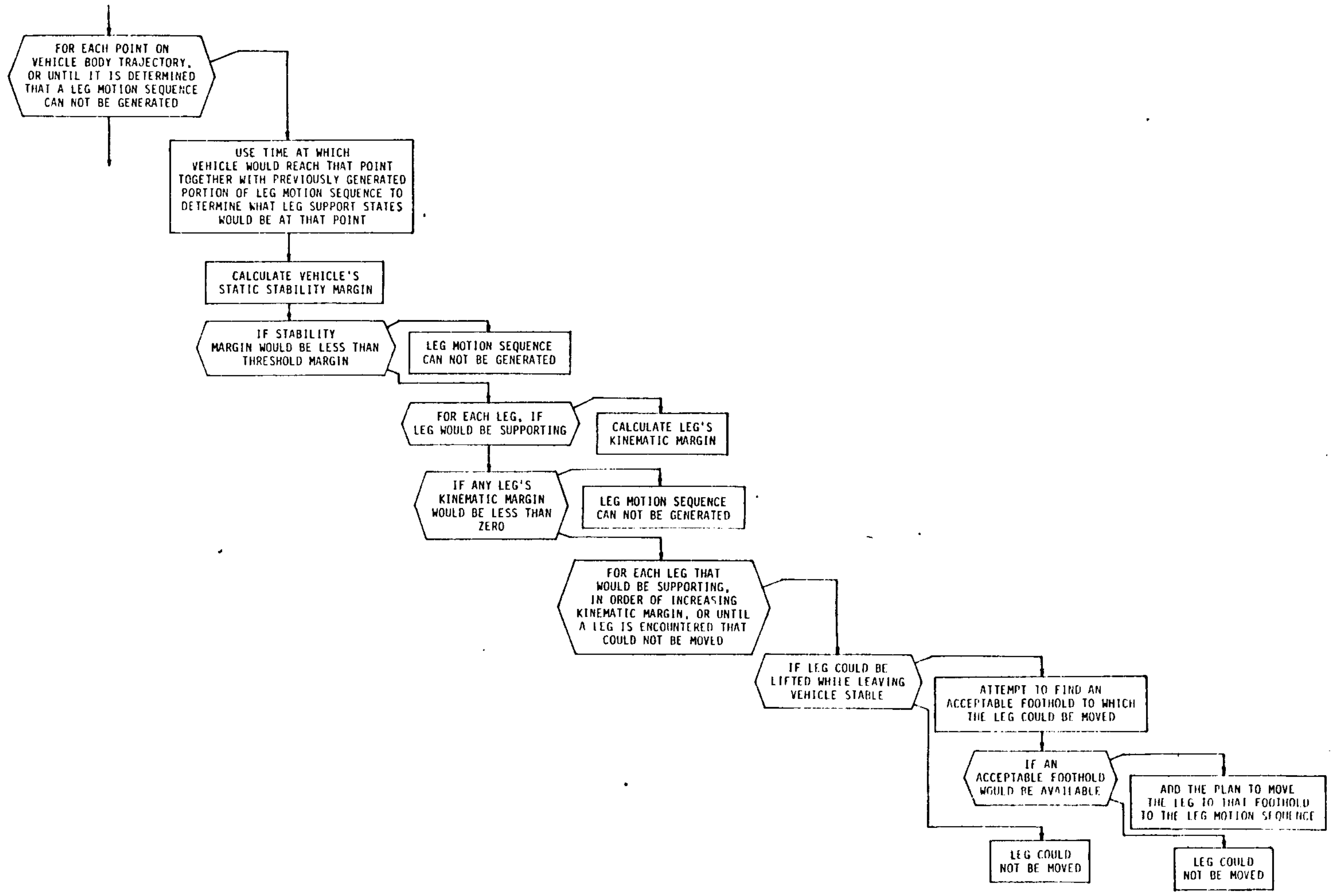

FOR EACH POINT ON VEHICLE BODY TRAJECTORY, OR UNTIL IT IS DETERMINED THAT A LEG MOTION SEQUENCE CAN NOT BE GENERATED
USE TIME AT WHICH VEHICLE WOULD REACH THAT POINT TOGETHER WITH PREVIOUSLY GENERATED PORTION OF LEG MOTION SEQUENCE TO DETERMINE WHAT LEG SUPPORT STATES WOULD BE AT THAT POINT
CALCULATE VEHICLE'S STATIC STABILITY MARGIN
IF STABILITY MARGIN WOULD BE LESS THAN THRESHOLD MARGIN
LEG MOTION SEQUENCE CAN NOT BE GENERATED
FOR EACH LEG, IF LEG WOULD BE SUPPORTING
CALCULATE LEG'S KINEMATIC MARGIN
IF ANY LEG'S KINEMATIC MARGIN WOULD BE LESS THAN ZERO
LEG MOTION SEQUENCE CAN NOT BE GENERATED
FOR EACH LEG THAT WOULD BE SUPPORTING, IN ORDER OF INCREASING KINEMATIC MARGIN, OR UNTIL A LEG IS ENCOUNTERED THAT COULD NOT BE MOVED
IF LEG COULD BE LIFTED WHILE LEAVING VEHICLE STABLE
ATTEMPT TO FIND AN ACCEPTABLE FOOTHOLD TO WHICH THE LEG COULD BE MOVED
IF AN ACCEPTABLE FOOTHOLD WOULD BE AVAILABLE
AND THE PLAN TO MOVE THE LEG TO THAT FOOTHOLD TO THE LEG MOTION SEQUENCE
LEG COULD NOT BE MOVED
LEG COULD NOT BE MOVED

FIGURE 3. VEHICLE LEG MOTION SEQUENCE GENERATION FLOW CHART

body at that point, together with the position of the vehicle body at that point in the body trajectory. If the vehicle would be statically unstable at that point, the generation of the leg motion sequence is aborted; if the vehicle would be stable, the motion sequence is aborted; if the vehicle would be stable, the motion sequence generation continues as described in the following paragraphs.

The next step in the iteration is to calculate, for those legs that would be supporting the vehicle, the legs' kinematic margins, which are the distances along the body trajectory over which the vehicle could move until the legs reach their kinematic limits. If any of the legs would have a kinematic margin of less than zero at the point on the trajectory that is being considered (that is, if any of the legs would be out of its limits when the vehicle was at that position), then the generation of the leg motion sequence is aborted.

Otherwise, the algorithm, beginning with the leg with the lowest kinematic margin and continuing to that with the highest, determines if any of the supporting legs could be moved. It does that by first evaluating whether the legs could be lifted while leaving the vehicle stable. If it could, the algorithm then attempts to find an appropriate foothold to which the leg could be moved; if it finds an acceptable foothold, it adds to the leg motion sequence the plan to move the leg to that foothold. If no foothold is found, or if the algorithm determined that the leg could not be lifted, the attempt to move a leg is terminated.

The procedure described in the proceeding paragraphs is performed at each point along the vehicle body trajectory. If at any point along the trajectory the procedure is aborted, the vehicle is not able to implement the accelerations that were used to generate the body trajectory. Otherwise, when the last iteration of the procedure at the last point on the trajectory is performed successfully, the leg motion sequence for the body trajectory is complete.

2.4 Guidance System Hardware

2.4.1 Overview

A diagram of the overall structure of the guidance system hardware and its internal and external communication channels is shown in Figure 4. It can be seen there that for its external communication the system receives information from the terrain-scanning system and both sends information to and receives it from the vehicle control system. As can also be seen in the figure, nearly all of the guidance system's internal inter-board communication takes place over the Intel Multibus, although in one case parallel data lines between boards are used.

Four of the six boards in the system are Intel iSBC 86/30 microprocessor boards, two of which contain an Intel 8087 numeric data processor as well as the Intel 8086 processor with which the boards are normally equipped. One of the other two boards in the system contains special-purpose circuitry designed to receive the data from the terrain-scanning system. The sixth board is a 512-kilobyte memory board in which the terrain elevation map is stored.

As shown in Figure 4, the six boards can be divided into three subsystems. One of the subsystems consists of a single processor which performs the tasks required for the communication with the vehicle control system. The other two subsystems perform the elevation map processing and the vehicle guidance processing; they communicate only through the terrain elevation map. The three subsystems are described in the following three parts of this section.

2.4.2 Elevation Map Subsystem

The elevation map subsystem receives terrain range data from the terrain scanner, converts those data to terrain elevations in Cartesian coordinates, and then stores the elevations in a terrain elevation map.

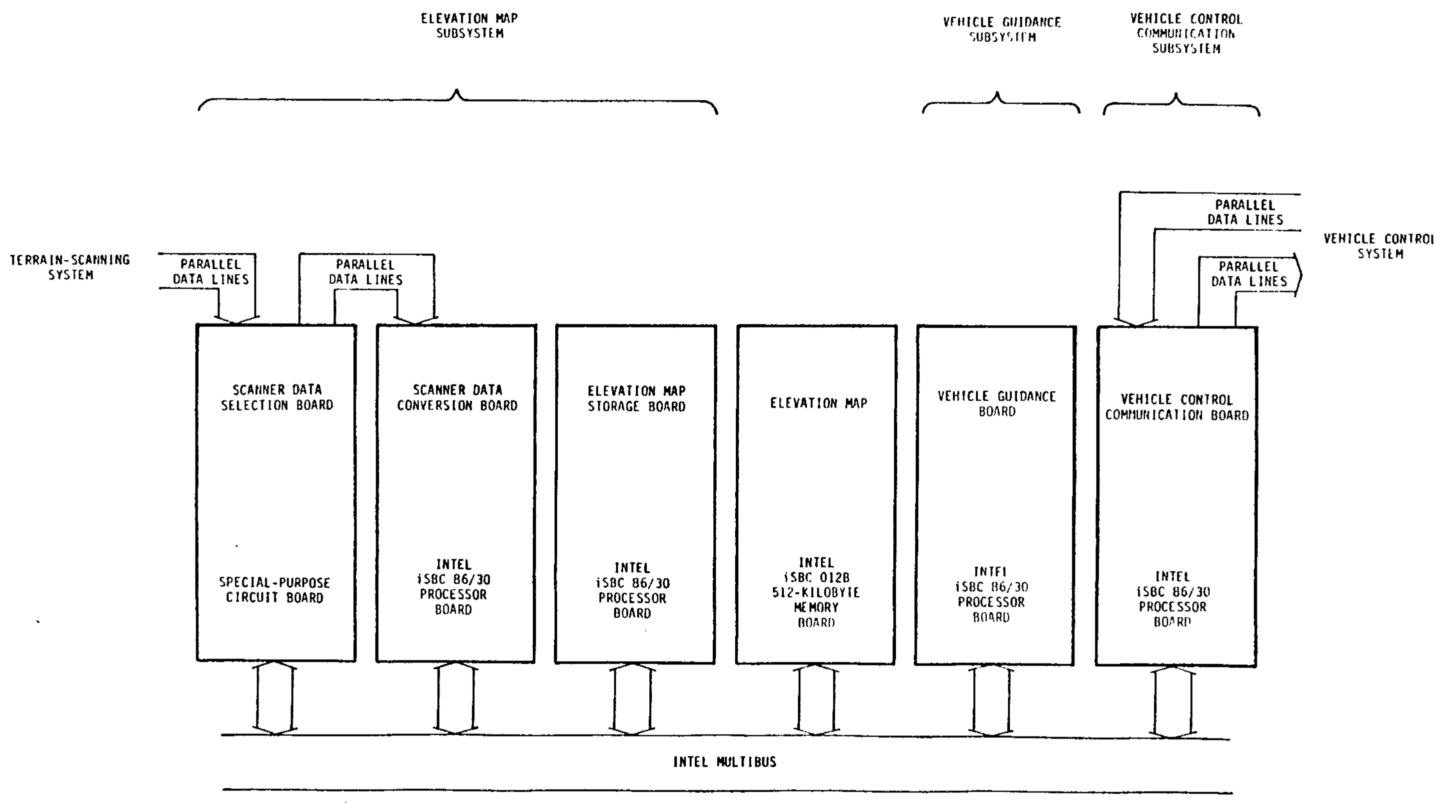

ELEVATION MAP SUBSYSTEM
VEHICLE GUIDANCE SUBSYSTEM
VEHICLE CONTROL COMMUNICATION SUBSYSTEM
TERRAIN-SCANNING SYSTEM
PARALLEL DATA LINES
PARALLEL DATA LINES
PARALLEL DATA LINES
PARALLEL DATA LINES
VEHICLE CONTROL SYSTEM
SCANNER DATA SELECTION BOARD
SPECIAL-PURPOSE CIRCUIT BOARD
SCANNER DATA CONVERSION BOARD
INTEL iSBC 86/30 PROCESSOR BOARD
ELEVATION MAP STORAGE BOARD
INTEL iSBC 86/30 PROCESSOR BOARD
ELEVATION MAP
INTEL iSBC 012B 512-KILOBYTE MEMORY BOARD
VEHICLE GUIDANCE BOARD
INTEL iSBC 86/30 PROCESSOR BOARD
VEHICLE CONTROL COMMUNICATION BOARD
INTEL iSBC 86/30 PROCESSOR BOARD
INTEL MULTIBUS

FIGURE 4. VEHICLE GUIDANCE SYSTEM COMPUTER CONFIGURATION

Two boards are used to perform the processing needed for the coordinate conversion and storage of the data so that the calculations for each range value can be performed very rapidly. However, even with the two processor boards, the scanner sends data faster than the calaulations can be performed, so a third board was designed and constructed to select only a given portion of the scanner data to pass to the processor boards.

Thus, the elevation map subsystem consists of one special-purpose circuit board and two Intel iSBC 86/30 processor boards. The operations of the three boards are described in the following paragraphs.

2.4.2.1 Scanner Data Selection Circuit. The scanner data selection board (the board layout and circuit diagram of which are shown in Appendices B and C, respectively) receives over parallel lines from the terrain-scanning system data consisting of the terrain range values and the line numbers in the raster-style scan for those range values. The board also has one memory location on the Multibus into which the scanner data conversion processor can write a number from one to fifteen. That number in the memory location is used by the circuit to determine what proportion of the range data should be passed to the data conversion processor: for the number n, every nth datum in every nth row of data is passed to the conversion processor.

In operation, the board receives every range datum from the scanner and then acknowledges that reception to the scanner. However, until a number is written into the board's memory, none of the data is passed on to the scanner data conversion processor. When a number is written into the memory, the board begins its normal operation, in which the data to be passed to the conversion processor are determined as described in the preceding paragraph. Those data that are to be passed are sent over parallel lines to the conversion processor board, which is described in the next section.

2.4.2.2 Scanner Data Conversion Processor. The scanner data conversion board, the computer program listings of which are shown in Appendix D, receives the scanner range data over parallel lines, as described in the preceding section. In addition, it receives frequent information over the Multibus from the vehicle control communication board concerning the current position and orientation of the vehicle. It uses that information as described in Section 2.3.2 to convert the scanner range data to terrain elevations. It then sends those elevation values over the Multibus to a buffer on the elevation map storage processor board which is described in the next section.

2.4.2.3 Elevation Map Storage Processor. The elevation map storage board, the program listings of which are shown in Appendix E, contains a buffer into which the scanner data conversion board writes terrain elevation values, as described in the preceding section. The only task of the map storage board is to store the elevation values at the proper locations (determined as described in Section 2.3.2) in the elevation map memory board. The elevation map can then be used by the vehicle guidance subsystem, which is described in the next section.

2.4.3 Vehicle Guidance Subsystem

The purpose of the single Itel iSBC 86-30 processor board that comprises the vehicle guidance subsystem is to generate, with the computer programs listed in Appendix F, the vehicle's body trajectories and leg motion sequences. For that purpose, it receives, over the Multibus from the vehicle control communication board, information concerning operator velocity requests, current vehicle body state (position and velocity), and the positions and support states of the legs. The guidance board uses that information, together with the elevation map on the memory board, to generate the body trajectories and leg motion sequences as described in Section 2.3.3. It stores the trajectories and motion sequences in a buffer on the board itself, where the information in them can be accessed by the vehicle control communication subsystem, which is described in the next section.

2.4.4. Vehicle Control Communication Subsystem

The vehicle control communication subsystem, which consists of one Intel iSBC 86/30 processor board using the programs listed in Appendix G, is responsible for all communications between the guidance system and the vehicle control system. The communication subsystem periodically receives, over parallel lines from the vehicle control system, the operator's vehicle velocity requests, the current vehicle body state (position and velocity), and the positions and support states of the vehicle's legs. It then passes all that information over the Multibus to the guidance subsystem. In addition, the communication board sends vehicle position information over the Multibus to the scanner data conversion board; since, to provide accurate conversion of the scanner data, the conversion board requires new vehicle position information more frequently than it is provided by the vehicle control system, the communication board interpolates between the vehicle positions provided by the control system by asking the vehicle velocity information that it also receives from the control system.

Finally, the communication board also provides the vehicle body and leg motion commands to the vehicle control system. It derives those commands from the body trajectories and leg motion sequences stored in the buffer on the vehicle guidance board, and it sends the commands over parallel data lines to the control system.

2.5 Conclusion

In summary, this portion of the report describes a computer system developed to enable a legged vehicle to walk over rough terrain. The system uses data from a terrain-scanning system, information from the vehicle's control system, and knowledge of the vehicle's capabilities and limitations to determine the body and leg motions required for the vehicle's locomotion over the terrain. The system has been extensively tested with the terrain scanner and with a breadboard version of the vehicle control computer, and it will soon be installed in the vehicle itself.

3.0 FOOT-LIFT SAFETY VALVE RESEARCH

3.1 Background and Valve Operation

In the development of any complex system such as the ASV, fail-safe measures must be incorporated into the system design to minimize or eliminate the possibility of a catastrophic system failure, i.e., one resulting in substantial damage to the machine or harm to the operator. Such failures could result from failure of one or more of the sensor systems, a computer malfunction, or unanticipated terrain.

These fail-safe mechanisms are primarily intended as a back-up to the first two safety mechanisms:

1) Proper operation and maintenance of the ASV.
2) Experimental procedures designed to minimize any risk to the machine or the operator.

3.2 Valve Operation

In discussion with program personnel at Ohio State University, it was agreed that some type of fail-safe mechanism should be designed into the hydraulic systems for the legs of the ASV. Two alternative approaches were considered. The first approach involved a safety valve that would connect the pressure side of the foot-lift circuit to the return side, in parallel with the actuator. Both sides of the actuator would, in effect, be connected together. As a result, there would be no pressure differential across the actuator, and the actuator would not support any load. The vehicle leg would, in effect, "go limp" and the vehicle, driven by its own weight, would do a "belly flop". A restriction in the safety valve would provide some control to this dropping action of the vehicle and would dissipate some of the vehicle's energy before it hit the ground. However, this would also reduce the speed of leg response.

A second approach was to install a similar safety valve that would connect the pressure side with the return side in place of the actuator. In this approach, the vehicle legs are "frozen" in place. This is accomplished by blocking off each end of the actuator, and preventing the flow of fluid out of either end. Each leg will act like a rigid structural member connected to the mainframe. This will probably result in a strong, jerking motion, and inertial forces may tend to roll the vehicle.

In some situations, the "belly flop" approach was felt to be the safest approach, while other situations seemed to require the second alternative. To accomodate this double requirement, a system was designed that was capable of providing either alternative. A schematic representation of this system is shown in Figure 5. If only the dump valve is actuated, the vehicle will drop. If both the dump and the block valves are actuated, the vehicle legs will be locked up.

Operation of the walking machine is so complex that the operator will not be able to react quickly enough to manually actuate the valves. Therefore, the system was designed for operation by the vehicle·computer through a established preprogrammed procedure. Depending on the circumstances, the computer may dump some of the legs and keep others stiff to control where and how the machine falls. This would be useful if it were necessary to avoid an object while falling. Computer control could also be used to prevent the walking machine from rolling over. The actual safety algorithms are to be developed by Ohio State University at a future date.

3.3 Valve Design

The safety valve was required to handle the full output flow of the pump (30 GPM). Since the hydrostatic circuit for each leg actuator is relatively short, the flow losses through the valve must be minimized to control the temperature rise of the hydraulic fluid. Flow losses through the valve were to be limited to approximately 30 psi. Finally, the overall package was to be as small as possible to allow it to fit into the leg assembly.

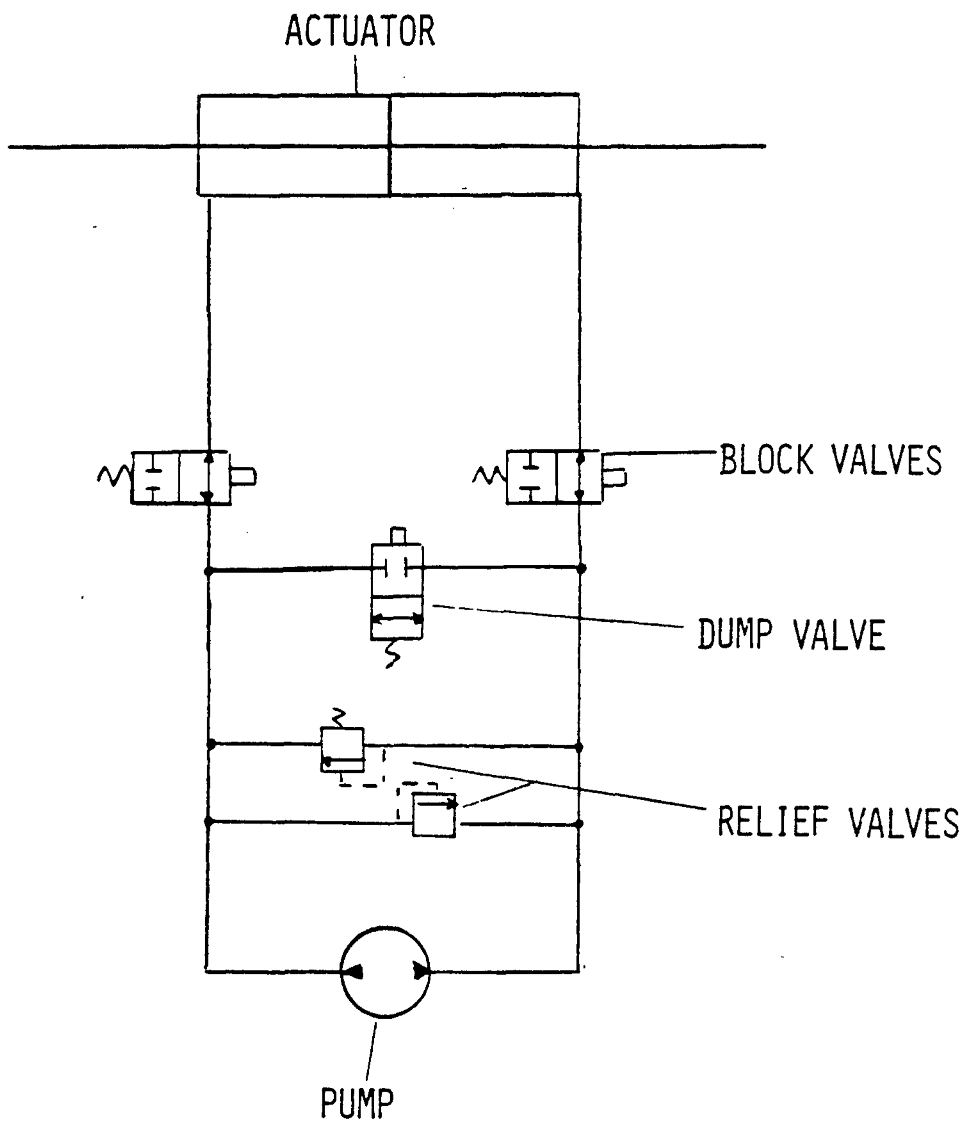


FIGURE 5. HYDRAULIC SCHEMATIC OF SAFETY VALVE CIRCUIT

Several different packaging alternatives were considered. One option was to mount a single valve directly on the pump output ports. A second option was to machine an internal valve in the crossover relief valve on the pump. The third alternative was a separate valve block with all three functions in it. The alternatives were evaluated, in conjunction with OSU, and the third option, the separate valve block, was selected.

3.3.1 Valve Actuation

Unless major problems with the system develop during testing, the actuation of the safety valve will be very infrequent. However, when the valve does operate, it must do so with high reliability. Solenoid-actuated devices were not felt to have sufficient reliability for this application. On the other hand electrically-actuated explosive device such as explosive bolts, have demonstrated high reliability, even after being installed for weeks or months. The major drawback with these devices is that they are not reuseable; a new one must be installed after each use. Any explosive actuator chosen must therefore be relatively inexpensive. Since explosive bolts cost in the range of 100 dollars each, they were considered to be too expensive for this application. A lower cost alternative, electrically-fired primers (Olin BWP-8-4-257W) were found to offer high reliability at a reasonable cost. These devices could easily be adapted to actuate the safety valve with a simple spring-activation technique.

3.3.2 Valve Design

The valve layout can be seen in Figure 6, a photograph of the valve. The valve body has three parallel spools running completely through the valve body. The center spool is the dump section and the two outside spools are the block sections of the valve. The three explosive cavities are at one end of the valve spool (bottom of picture) and their bolt-on flanges are shown. At the other end of the valve

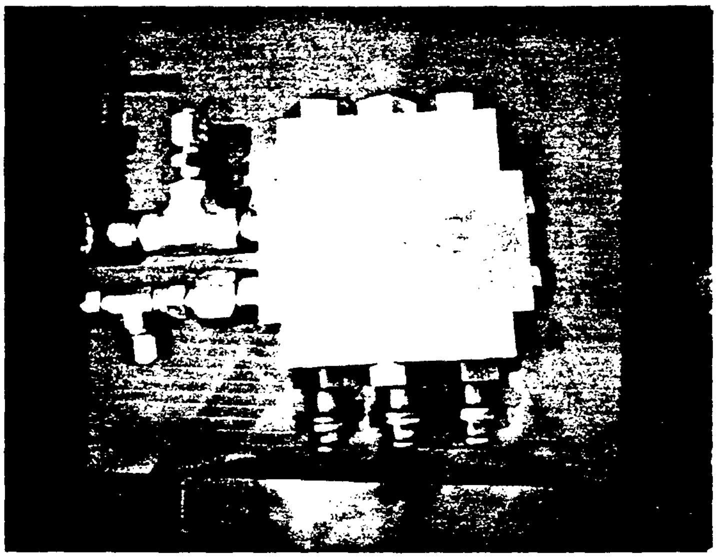


FIGURE 6. SAFETY VALVE

(top of photo), there is a hex plug which contains the compression spring. Access can be gained to the spool through either end of the valve. The side of the valve contains the hydraulic connection ports. The ports are sized for a SAE-10 straight threaded connection. The connections shown on the left are for a pump-to-actuator port connection. The right side would be the same except that the pump and actuator ports would be switched. The electrical connections are the thin wires visible underneath the explosive cavities. Only one common ground connection is necessary for all three explosive housings.

3.3.3 Operating Pressure

In the current system concept, the leg actuator system pressure will range from a minimum of 150 psi to a maximum of 4000 psi. During normal operation the pressure will be considerably less than the maximum (~400-500 psi). However, pressure transients may occur which result in pressure spikes higher than the 4000 psi maximum. Since the seal on the sliding spool shaft must be capable of handling this pressure without extruding, back-up rings were placed on both sides of the seal.

3.3.4 Operating Time

The valve was also required to have a quick operating time so that the safety procedure could be actuated with enough time to be effective. OSU estimated that the required time of operation was 50 milliseconds (msec) for complete shifting, once the electrical signal had been applied. This required the use of a stiff spring to accelerate the spool fast enough. It also required that the explosive primer operate quickly. (The published time for primer operation is less than 2 msec.)

3.3.5 Spool Clearance

In order for the safety valve to operate with high reliability it is necessary to ensure that the valve spool shifts. The major cause of spool hang-up is radial forces applied to the spool due to uneven pressure distribution. These radial forces cause friction between the valve wall and the spool preventing the spool from moving. The primary way to avoid these forces is to eliminate any pressure drops acting on on the side of the spool. To accomplish this, the counterbores around the ports were designed as large as possible so that the pressure around the spool is equal in all radial directions. Equal pressure around the spool avoids any radial forces caused by fluid flow.

Although it is important that there be enough valve body/spool clearance to allow free movement of the spool, the larger the clearance, the greater the valve leakage. The leakage rate must be minimized to assure proper system performance. The leakage rate of the spool can be determined by the equation:

$$Q = \pi r\, c^3\, (P_1 - P_2)/2\mu L$$

where

Q = valve leakage rate
c = clearance
r = radial bore
μ = viscosity
$P_1 - P_2$ = pressure drop across the spool
L = length of spool land

This is the average rate of flow lost through the dumping section of the safety valve during normal operation. It is small enough to assure proper system performance.

Once the valve has been actuated (in the "locked leg" mode), fluid will leak across the block valves. Since these valves will normally not see a 4000 psi pressure differential across them, the leakage rate will be less than 0.091 GPM. As the blocking valves leak, the legs will slowly creep and the vehicle body will drop. However, the rate will be quite slow under these circumstances and is considered acceptable.

The forces acting along the axis of the spool control the shifting of the valve spool. In order to shift the spool, there must be a net force in the direction of shifting. The valve is designed so that a force due to an area differential in the spool will act in conjunction with a spring to shift the spool. Opposing this movement is spool-wall friction, rod-seal friction and flow losses moving fluid through the spool. Calculation of these forces (Appendix H) indicates that, even under high pressures, the net force is always positive in the shifting direction.

3.4 Valve Operation

Under normal operation of the valve, fluid from the pump enters the valve port and passes through four 1/2-inch holes in one of the outside spools (depending on the direction of operation). The fluid then flows axially along the 3/4-inch diameter bore in the center of the spool. At the other end of the spool, the fluid exits through a second set of ½-inch diameter holes. (For purposes of calculation, then, the valve may be treated as two orifice valves, each with four 1/2-inch diameter holes and a section of 3/4-inch diameter pipe approximately 4 inches long.) The return flow passes through the other outside spool in a similar manner. The center spool normally is blocked (the offset holes in the center spool are covered by the valve body lands) so that no flow passes through it.

To activate the valve, a electrical signal from the computer/ electronics system is applied to the explosive actuator. A signal pulse of 24 volt D.C. at the actuator of at least 2 msec in duration is required. Voltage drops from the power source to the valve are not included in this value. The electrical connection to the valve is through three 22 gauge wires. The valve body is grounded to the leg frame for the other electrical connection.

Once the electrical signal is given to the valve, the explosive actuator will detonate. Gases produced by the explosion are contained in the ceramic tube, where they build up pressure, rupturing the tube. The ceramic tube is completely destroyed by the explosion, freeing the spring to push the spool to its offset position. As each of the outside block spools shifts, the radial holes in it are covered by the valve body lands, and flow is shut off. The spool, when completely shifted, has moved 5/8 inch. The length of the spring has been designed to ensure that there is spring pressure holding the spool in the shifted position. The operation of the center dump spool is identical except that the radial holes go from a covered to an uncovered land position, allowing flow.

3.5 Valve Refurbishment

The valve body contains an explosive cavity that was designed for two purposes: to contain the fragments from the explosion and to facilitate the installation of the next primer assembly. The tube around the primer assembly contains the metal and ceramic fragments produced during the explosive activation. Clearance with the outside tube is sufficient to allow the escape of the explosion gases. The explosion cavity and washers also serve to align the primer and ceramic tube so that the compressive force is transmitted directly down the spool axis. The outer casing around the explosion cavity provides a guide when pushing the assembly to its closed position.

To install the new explosive assembly after a valve actuation the following procedures are followed: take off the explosive housing by removing the four bolts. The brass plug, brass tube holder and old explosive should be removed. Remove the old ceramic tube fragments from the explosive cavity. To make up the next explosive assembly, take the brass plug and the ceramic tube and glue them together so that the ceramic tube is blocked on one end and is centered on the plug. Next, remove the old explosive from its centering ring and replace it with a new explosive. Place the brass tube holder over the explosive and make sure the explosive (red color) can be seen through the ceramic tube holder. Place the ceramic tube holder and explosive into the explosive cavity and feed the wire out through the hole provided for it.

The bottom of the explosive should touch the bottom of the explosive cavity. Next, place the ceramic tube/brass plug into the ceramic tube holder. The brass plug should sit flush with the end of the explosive cavity.

The explosive cavity is now ready for reinstallation. Place the explosive cavity over the end of the spool shaft. The brass plug should be centered on the spool shaft. The system pressure must be relieved before reinstallation or the spool shaft will be hard to move. Push on the explosion cavity slowly until the explosion cavity flanges meet and then reinstall the four bolts. The electrical connection must be completed before the valve is ready to fire.

3.6 Testing

The testing of the valve was done to confirm its operational characteristics. The test was performed using a hydraulic power supply with a 5 GPM output and a maximum of 3200 psi. After operational tests the unit was tested to 4000 psi to check for leakage.

Testing showed that there was no leakage found with the power unit under a static 3200 psi pressure. In addition the spools were shifted manually under pressure to see if the seals would work dynamically and still no leakage was found. Even after explosive actuation with quick seal movement there was no leakage. There was no sign of seal extrusion when checked after testing.

In order to measure the pressure drop through the valve block section when there is free flow (un-actuated), the pressures above and below the valve were measured. These pressures were 76 and 72 psi respectively. There is also a tube fitting restriction to be accounted for, which produces, according to calculation, over 3 psi in pressure drop. Therefore, at a 5-GPM flow rate, the valve's pressure drop is less than 1 psi. Because pressure drop increases by the square of the flow, then at a 30-GPM flow the pressure drop would be less than 36 psi.

The operational test consisted of measuring the shifting time of the valve spools. The blocking valve was tested by allowing a flow through the valve and applying a downstream pressure with a restriction. The pressure was measured using a pressure transducer located on the downstream side of the valve. When the blocking valve was triggered, it shifted, shutting off flow to the restriction. The loss of flow will cause the pressure to drop at the pressure transducer. The pressure vs. time and the trigger pulse vs. time were measured for several different pressures. The shift time is the difference between the trigger pulse and when the pressure reads zero. Refer to Figure 7 for the hydraulic schematic of the test set-up.

The results of two trials on the blocking valve are shown in Figure 8. The traces show that the valve has a shifting time of less than 12 msec. It is difficult to determine from the trigger pulse trace where the explosive primer actuates. However, it seems from the trace of pressure that the spool itself takes only about 3 msec to shift once it starts. The explosive actuator in the trial was actuated by a 6 V.D.C. battery. Using a high voltage source such as 24 V.D.C., could actuate the explosive quicker and hence shorten the total valve shifting time. The trigger pulse also shows voltage "bounce" which is probably caused by mechanical vibrations in the triggering switch. Eliminating this bounce could shorten the shifting time.

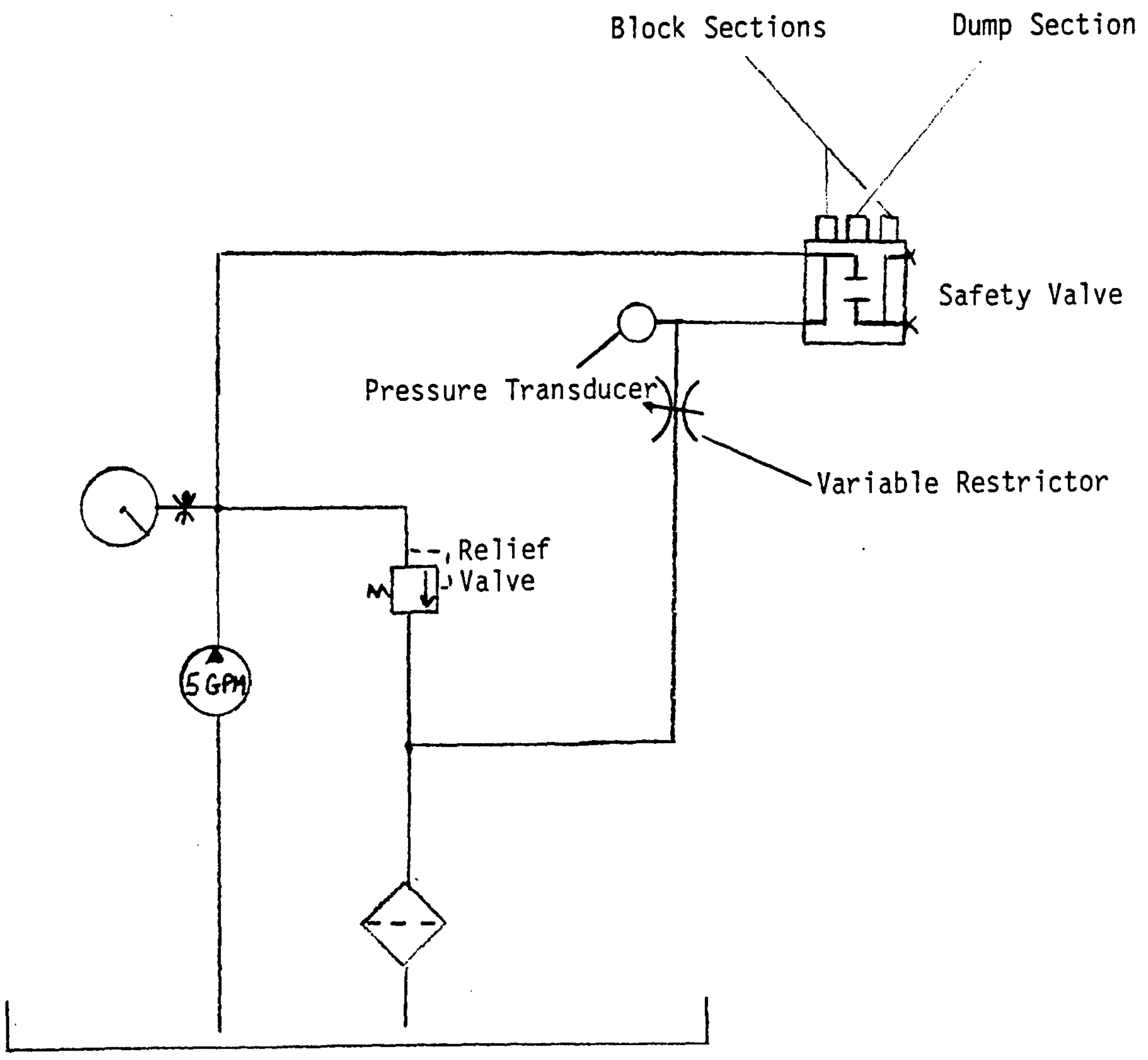
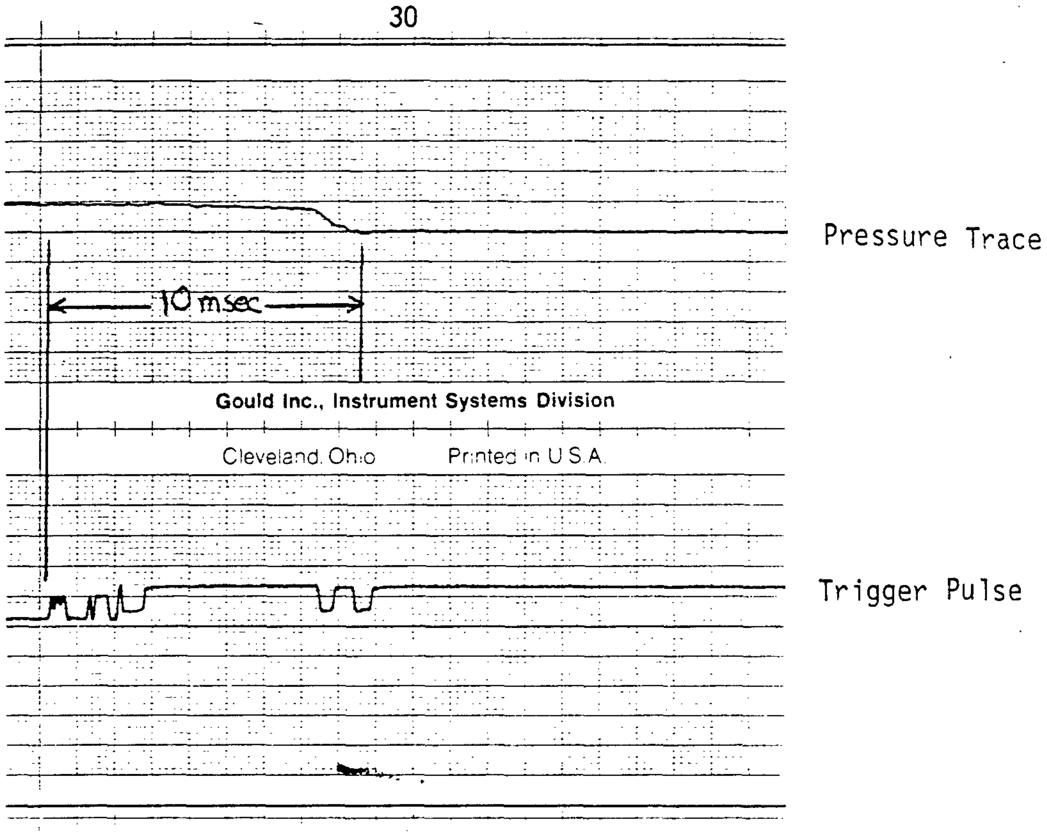
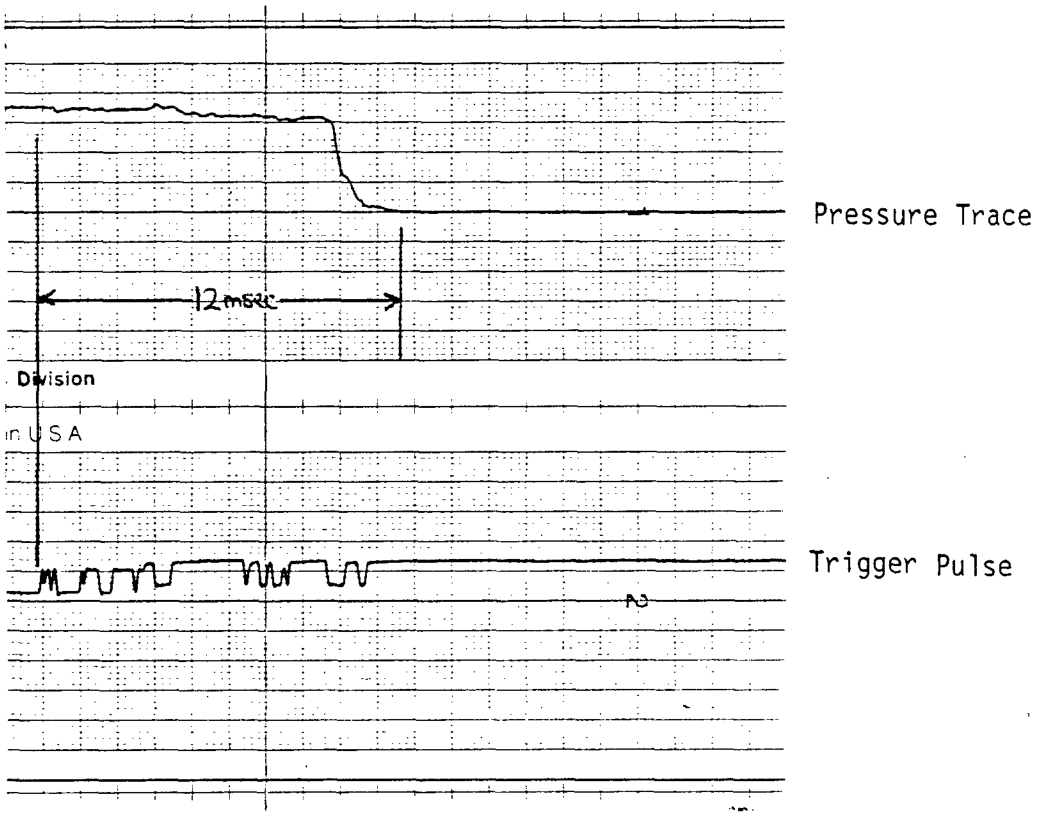


FIGURE 7. HYDRAULIC SCHEMATIC FOR TEST SET-UP

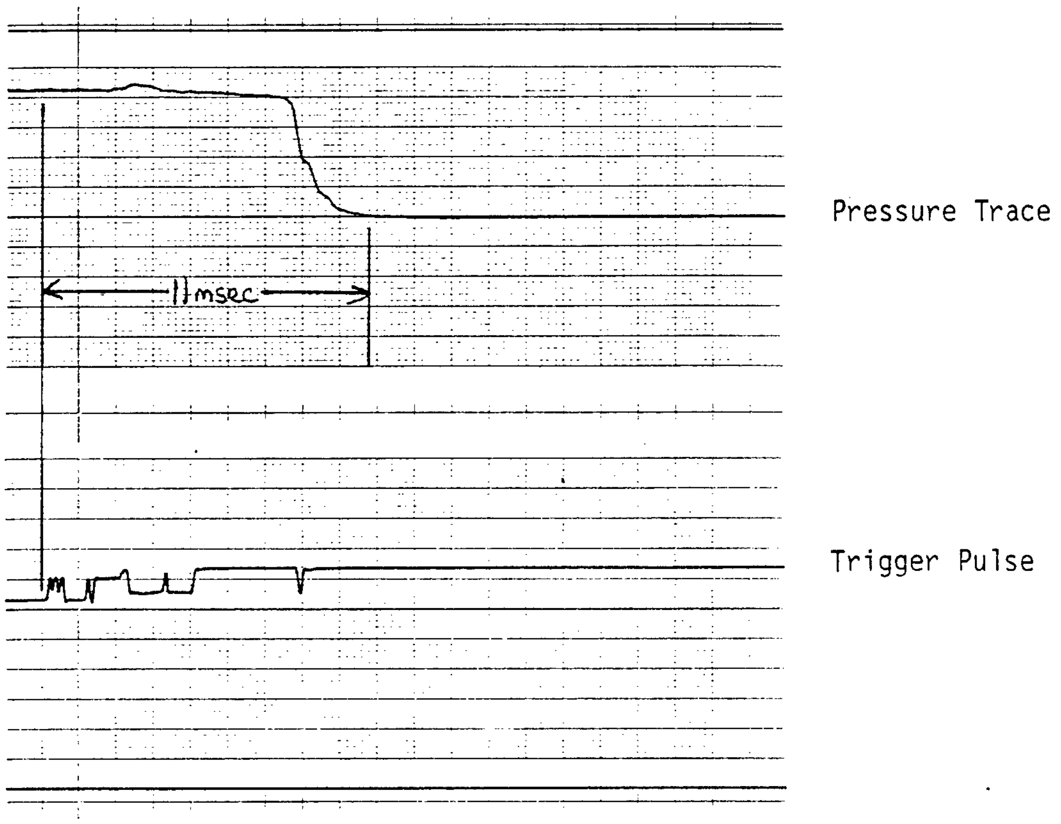


Trial 1 - 200 psi back pressure



Trial 2 - 800 psi back pressure

FIGURE 8. OPERATIONAL TEST RESULTS



Trial 3 - 1000 psi back pressure

FIGURE 8. OPERATIONAL TEST RESULTS

APPENDIX A

FLOW CHART CONVENTIONS

FLOW CHART CONVENTIONS

The purpose of this appendix is to describe the conventions used in the flow charts (Figures 1, 2, and 3) in this report, where those conventions differ from those commonly used in flow charts.

The charts consist basically of vertical sequences of blocks, with control passing sequentially down through the sequences. However, interspersed in those sequences are decision (hexagonal) blocks, which require the execution of subsequences of blocks located to the right of the decision blocks. When the bottom of a vertical sequence of blocks is reached, control returns to the decision block from which that sequence began.

As to the decision blocks themselves, when the condition in the block is described by a FOR, WHILE, or UNTIL phrase, the subsequence to the right is executed, respectively, FOR the conditions stated, WHILE the condition stated is true, or UNTIL the condition stated is true. If the decision block contains an IF phrase, the subsequence indicated by the arrow exiting from the upper right of the block is executed IF the condition is true. If there is a subsequence indicated by an arrow exiting from the lower right of the block, that subsequence is executed IF the condition is false.

Finally, rectangular blocks bounded by thick lines indicate sequences described by other flow charts.

APPENDIX B

SCANNER DATA SELECTION BOARD LAYOUT

The following page shows the component layout of the scanner data selection board. (Component definitions and schematic are given in Appendix C.)

LA22P1

4608 PLUGBORD – COMPONENT SIDE LAYOUT PAPER

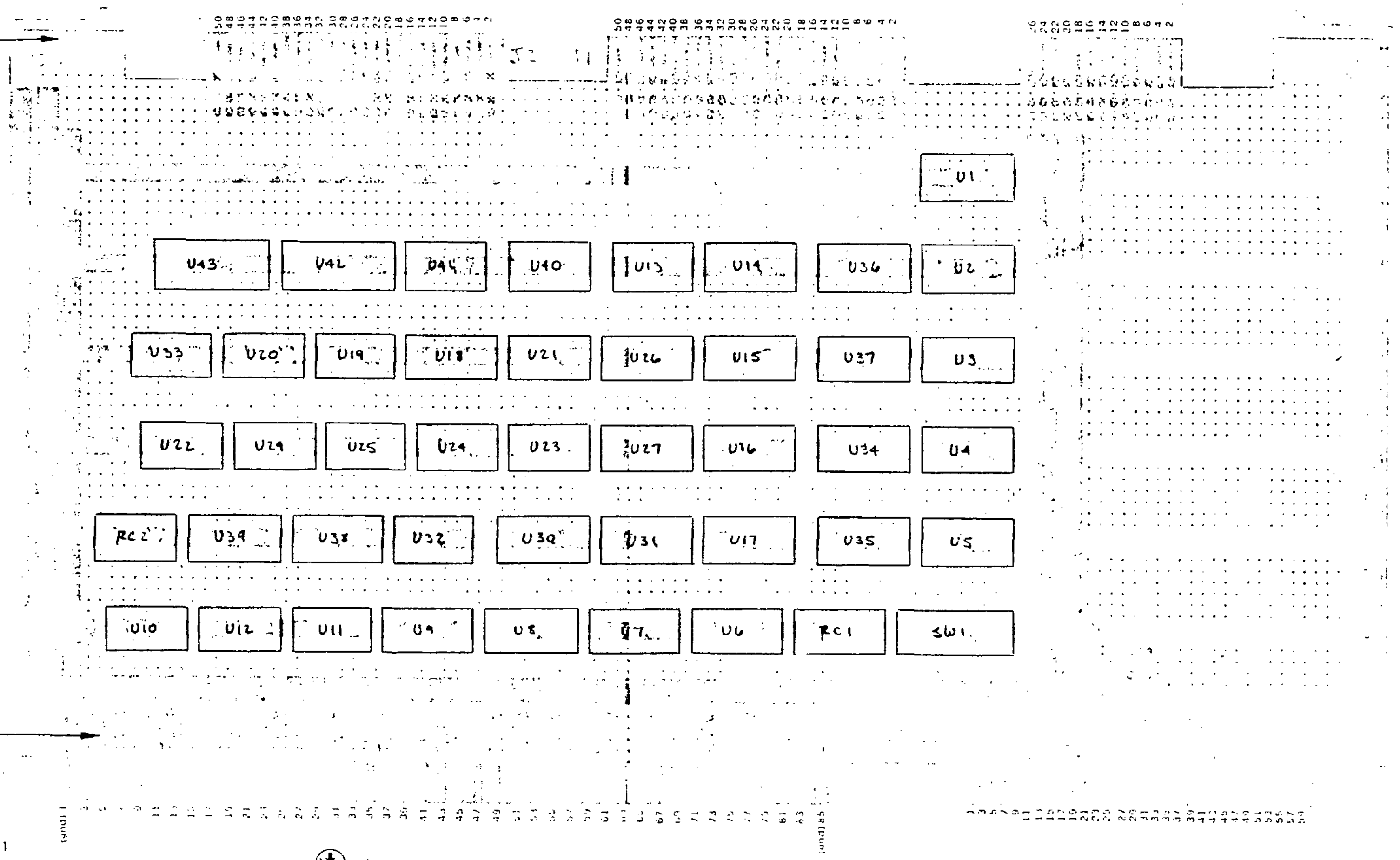
Approximate area required for Vector HA 9 extractor

APPLICATION: Intended for use in non-hostile environments up to 200 volts RMS or 300 volts DC.

NOTICE. Where tin-coated circuitry exists a small percentage of the holes may have solder blockage. This is a light "skin" easily penetrated by component leads. In some cases a soldering iron may be required.

Floating pads indicate location of connector contact pads on opposite side of board.

CAUTION: In any in-ass contact area on either side of Plugbord use only those holes having pads. Holes without pads may have insufficient clearance to adjacent circuitry and using them could cause shorting.



Vector DIP Plugbord pattern .042" x 0.1" spaced holes LA22P1 layout paper.

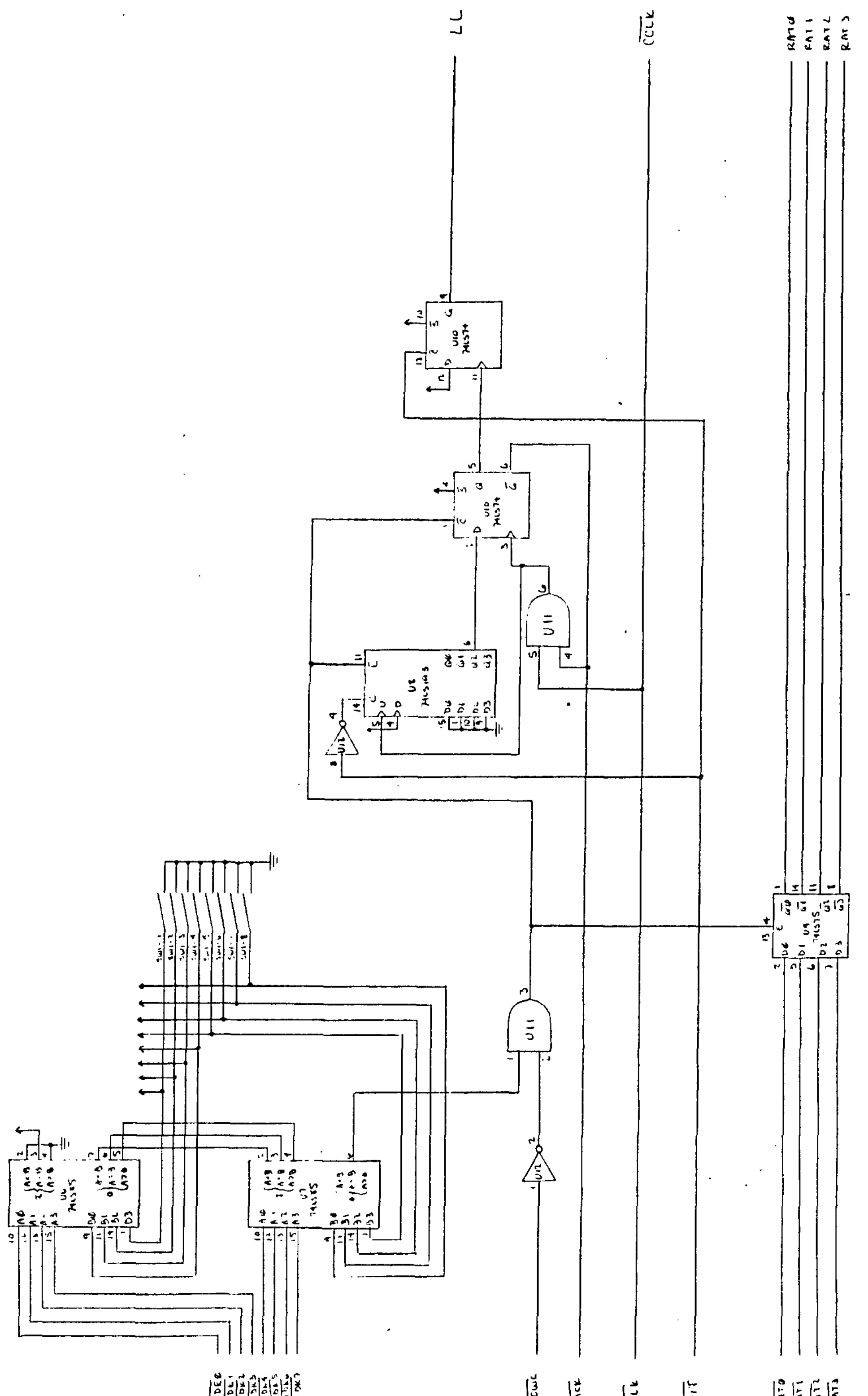
VECTOR ELECTRONIC COMPANY, INC. 12460 Gladstone Avenue, Sylmar, CA 91342

APPENDIX C

SCANNER DATA SELECTION BOARD SCHEMATIC

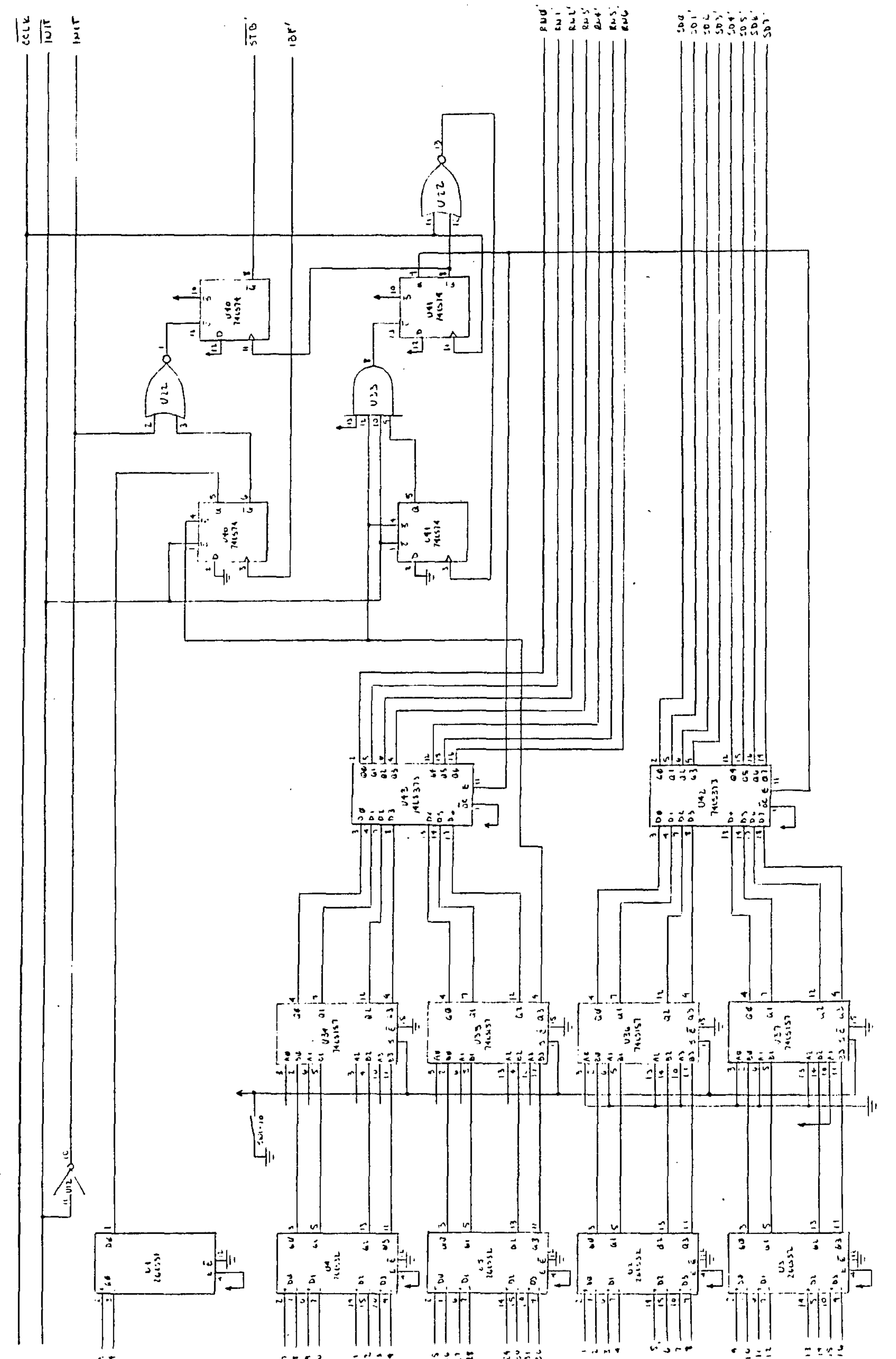
The following three pages give a schematic diagram of the scanner data selection board. (The component layout for the board is shown in Appendix B.)

C-1



LL
CCLK
U10 74LS74
U8 74LS193
U11
U12
U6 74LS85
U7 74LS85
U9 74LS75
IOWC
XACK
CCLK
INIT
MULTIBUS

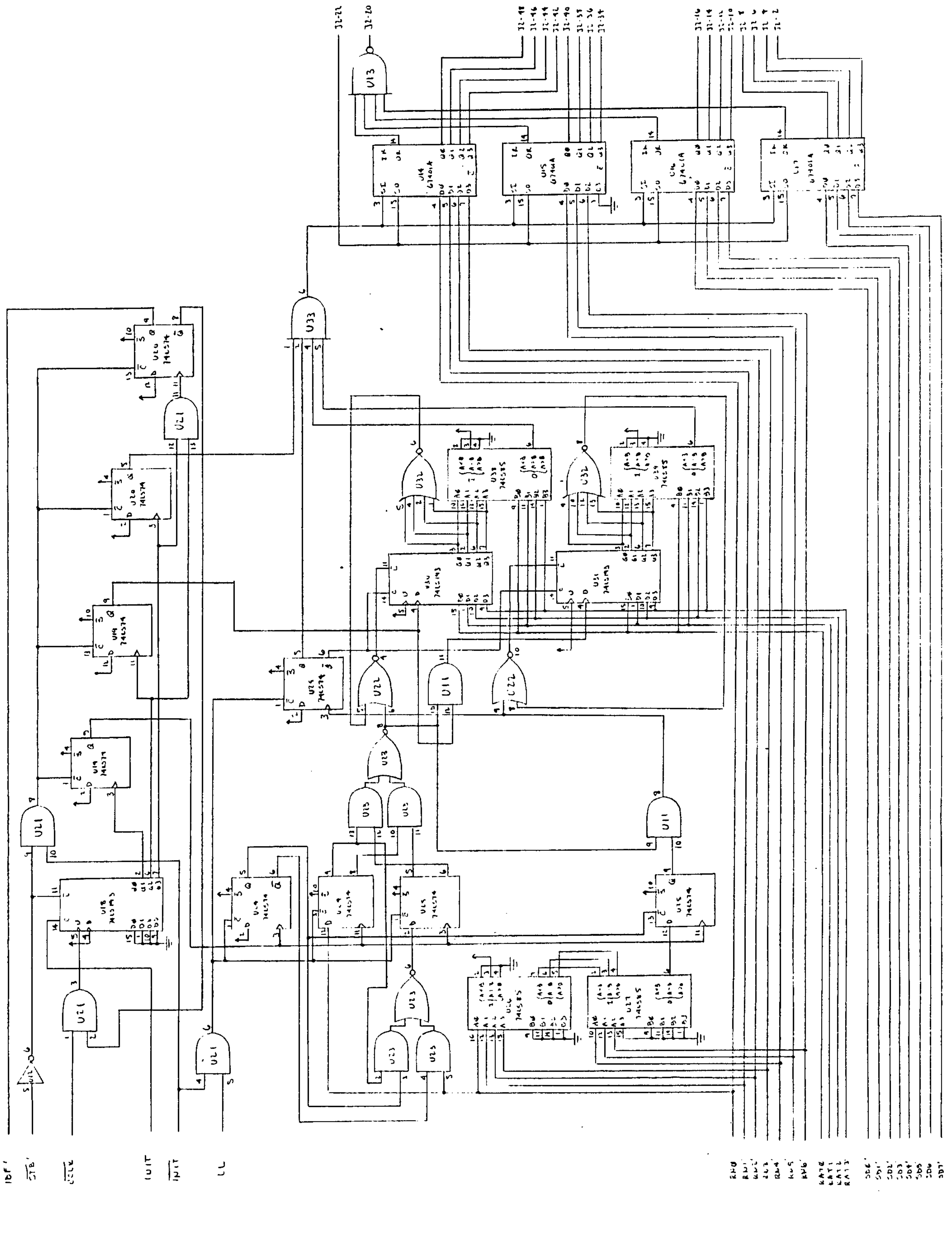
C-2



TERRAIN - SCANNING SYSTEM

C-3

SCANNER DATA CONVERSION BOARD



APPENDIX D

SCANNER DATA CONVERSION BOARD PROGRAM LISTING

```
MODULE ScannerRangeDataConversion;

PROGRAM ScannerRangeDataConversion (Input, Output);

  CONST

    Pi = 3.141592654;

    Real1Integer = 63;
    Real1IntegerPlus1 = 64;

  TYPE

    Bits = (Bit0, Bit1, Bit2, Bit3, Bit4, Bit5, Bit6, Bit7);
    SetOfBits = SET OF Bits;

    Byte =
    RECORD
      CASE Integer OF
        0: (Chrctr: Char);
        1: (BitSet: SetOfBits)
    END;

    TwoCharInteger =
    RECORD
      CASE Integer OF
        0: (LoChar,
            HiChar: Char);
        1: (IntVal: Integer)
    END;

    UnitVectorCompIntegerIndexedArray =
    ARRAY [-Real1IntegerPlus1..Real1Integer] OF Integer;

    Crd = (X, Y, Z);

    TransMatrixInteger =
    RECORD
      Rotat: ARRAY [Crd] OF ARRAY [Crd] OF Integer;
      Trans: ARRAY [Crd] OF Integer
    END;

    BooleanPtr =
    RECORD
      CASE Integer OF
        0: (AbsAddr: ^Boolean);
        1: (OffAddr,
            SegAddr: Integer)
    END;

    IntegerPtr =
    RECORD
      CASE Integer OF
        0: (AbsAddr: ^Integer);
        1: (OffAddr,
            SegAddr: Integer)
    END;

    UnitVectorCompIntegerIndexedArrayPtr =
    RECORD
      CASE Integer OF
        0: (AbsAddr: ^UnitVectorCompIntegerIndexedArray);
        1: (OffAddr,
            SegAddr: TwoCharInteger)
    END;

    Brds12CommBuffer =
    RECORD
      ScnrioEarthTransMatrix: TransMatrixInteger;
      NewData: Boolean
    END;

    Brds12CommBufferPtr = ^Brds12CommBuffer;

    Brds12CommBufferPtrPtr =
    RECORD
      CASE Integer OF
        0: (AbsAddr: ^Brds12CommBufferPtr);
        1: (OffAddr,
            SegAddr: Integer)
    END;

  VAR

    Brd2InOperationPtr: BooleanPtr;
    Brds12CommInptBufferPtrFromBrd2Ptr,
    Brds12CommIdleBufferPtrFromBrd2Ptr,
    Brds12CommOtptBufferPtrFromBrd2Ptr,
    Brds12CommInptBufferPtrFromBrd1Ptr,
    Brds12CommIdleBufferPtrFromBrd1Ptr,
    Brds12CommOtptBufferPtrFromBrd1Ptr: Brds12CommBufferPtrPtr;
    Brds12CommTempBufferPtr: Brds12CommBufferPtr;
    Brds12CommIdleBufferBusyFromBrd2Ptr,
    Brds12CommIdleBufferBusyFromBrd1Ptr: BooleanPtr;
    PortCByte: Byte;
    RangeDataConversionToBegin: Boolean;
    RangeDataConversionInProcess: Boolean;
    ScanLineNumber: TwoCharInteger;
    PrevScanLineNumber: Integer;
    NextScanLineNumber: TwoCharInteger;
    ScanPointNumber: TwoCharInteger;
    RangeDatum: TwoCharInteger;
    ScannerPosX,
    ScannerPosY,
    ScannerPosZ: Integer;
    ScanPtDispXElCosProdFact,
    ScanPtDispYElCosProdFact,
    ScanPtDispZElCosProdFact,
    ScanPtDispXElSinProdFact,
    ScanPtDispYElSinProdFact,
    ScanPtDispZElSinProdFact,
    ScanPtDispXAzCosProdFact,
    ScanPtDispYAzCosProdFact,
    ScanPtDispZAzCosProdFact,
    ScanPtDispXAzSinProdFact,
    ScanPtDispYAzSinProdFact,
    ScanPtDispZAzSinProdFact: Integer;
    ElCosProdArrayPtr,
    ElSinProdArrayPtr,
    AzCosProdArrayPtr,
      AzSinProdArrayPtr,
      ScanPtDispArrayPtr:
      UnitVectorCompIntegerIndexedArrayPtr;
      ScanPtPosDatumBufferFullPtr,
      ScanPtPosDatumBufferFullOnBrd3Ptr: BooleanPtr;
      ScanPtPosXPtr,
      ScanPtPosYPtr,
      ScanPtPosZPtr: IntegerPtr;
      ScanPtPosX,
      ScanPtPosY,
      ScanPtPosZ: Integer;
      PrevRangeDataArray: ARRAY [0..31] OF Integer;
      ScanPointNumberRangeDataAcceptable: ARRAY [0..31] OF Boolean;
      ArrayIndex1,
      ArrayIndex2: Integer;
      ArrayIndex2TwoChar: TwoCharInteger;
      CrdIndex,
      CrdIndex1,
      CrdIndex2: Crd;


  BEGIN

    DisableInterrupts;

    Brd2InOperationPtr. SegAddr := 8190;
    Brd2InOperationPtr. OffAddr :=    0;

    Brd2InOperationPtr. AbsAddr^ := False;

    Brds12CommIdleBufferBusyFromBrd2Ptr. SegAddr := 8189;
    Brds12CommIdleBufferBusyFromBrd2Ptr. OffAddr :=    1;
    Brds12CommIdleBufferBusyFromBrd1Ptr. SegAddr := 8189;
    Brds12CommIdleBufferBusyFromBrd1Ptr. OffAddr :=    0;

    Brds12CommIdleBufferBusyFromBrd2Ptr. AbsAddr^ := False;

    Brds12CommOtptBufferPtrFromBrd2Ptr. SegAddr := 8188;
    Brds12CommOtptBufferPtrFromBrd2Ptr. OffAddr :=    8;
    Brds12CommIdleBufferPtrFromBrd2Ptr. SegAddr := 8188;
    Brds12CommIdleBufferPtrFromBrd2Ptr. OffAddr :=    4;
    Brds12CommInptBufferPtrFromBrd2Ptr. SegAddr := 8188;
    Brds12CommInptBufferPtrFromBrd2Ptr. OffAddr :=    0;
    Brds12CommOtptBufferPtrFromBrd1Ptr. SegAddr := 8187;
    Brds12CommOtptBufferPtrFromBrd1Ptr. OffAddr :=    8;
    Brds12CommIdleBufferPtrFromBrd1Ptr. SegAddr := 8187;
    Brds12CommIdleBufferPtrFromBrd1Ptr. OffAddr :=    4;
    Brds12CommInptBufferPtrFromBrd1Ptr. SegAddr := 8187;
    Brds12CommInptBufferPtrFromBrd1Ptr. OffAddr :=    0;

    ScanPtPosDatumBufferFullPtr. SegAddr := 8191;
    ScanPtPosDatumBufferFullPtr. OffAddr :=    0;

    ScanPtPosDatumBufferFullOnBrd3Ptr. SegAddr := -6145;
    ScanPtPosDatumBufferFullOnBrd3Ptr. OffAddr :=     0;

    ScanPtPosXPtr. SegAddr := -6145;
    ScanPtPosXPtr. OffAddr :=     2;

    ScanPtPosYPtr. SegAddr := -6145;
    ScanPtPosYPtr. OffAddr :=     4;
```

```
ScanPtPosZPtr. SegAddr := -6145;
ScanPtPosZPtr. OffAddr :=      6;

Outbyt (0CFH, Chr (086H));

ElCosProdArrayPtr. SegAddr. IntVal := 2037;
ElCosProdArrayPtr. OffAddr. LoChar := Chr (000H);
ElSinProdArrayPtr. SegAddr. IntVal := 2549;
ElSinProdArrayPtr. OffAddr. LoChar := Chr (000H);

AzCosProdArrayPtr. SegAddr. IntVal := 3061;
AzCosProdArrayPtr. OffAddr. LoChar := Chr (000H);
AzSinProdArrayPtr. SegAddr. IntVal := 3573;
AzSinProdArrayPtr. OffAddr. LoChar := Chr (000H);

ScanPtDispArrayPtr. SegAddr. IntVal := 4065;
ScanPtDispArrayPtr. OffAddr. LoChar := Chr (000H);

ScanLineNumber. HiChar := Chr (000H);
RangeDatum. HiChar := Chr (000H);

RangeDataConversionToBegin := False;
RangeDataConversionInProcess := False;

PrevScanLineNumber := -1;
ScanPointNumber. IntVal := 0;

REPEAT
UNTIL Brd2InOperationPtr. AbsAddr^;

writeLn ('Board 2 is in operation.');

OutByt (0E0H, Chr (8));

WHILE True DO
BEGIN

  PrevScanLineNumber := ScanLineNumber. IntVal;

  REPEAT

    InByt (0CCH, PortCByte. Chrctr)

  UNTIL (PortCByte. BitSet * [Bit1]) <> [];

  InByt (0C8H, ScanLineNumber. LoChar);

  ScanLineNumber. IntVal := (255 - ScanLineNumber. IntVal) DIV 8;

  InByt (0CAH, RangeDatum. LoChar);

  IF RangeDataConversionToBegin
  THEN
  ScanPointNumber. IntVal := ScanPointNumber. IntVal + 1;

  IF RangeDataConversionInProcess
  THEN
  BEGIN

    IF ScanPointNumber. IntVal = 0
  THEN
  IF ScanLineNumber. IntVal <> NextScanLineNumber.IntVal
  THEN
  WriteLn ('#');

  IF ScanLineNumber. IntVal = 0
  THEN
  ScanPointNumberRangeDataAcceptable [ScanPointNumber. IntVal] := True
  ELSE
  IF ScanPointNumberRangeDataAcceptable [ScanPointNumber. IntVal]
  THEN
  ScanPointNumberRangeDataAcceptable [ScanPointNumber. IntVal] := NOT
  ((RangeDatum. IntVal - PrevRangeDataArray [ScanPointNumber. IntVal]) <
  -16);

  PrevRangeDataArray [ScanPointNumber. IntVal] := RangeDatum. IntVal;

  IF ScanPointNumberRangeDataAcceptable [ScanPointNumber. IntVal]
  THEN
  BEGIN

    AzCosProdArrayPtr. OffAddr. HiChar := ScanPointNumber. LoChar;
    AzSinProdArrayPtr. OffAddr. HiChar := ScanPointNumber. LoChar;

    ScanPtDispArrayPtr. OffAddr. HiChar := RangeDatum. LoChar;

    ScanPtPosX := ScannerPosX + ScanPtDispArrayPtr. AbsAddr^ [
    AzCosProdArrayPtr. AbsAddr^ [ScanPtDispXAzCosProdFact] +
    AzSinProdArrayPtr. AbsAddr^ [ScanPtDispXAzSinProdFact]];

    ScanPtPosY := ScannerPosY + ScanPtDispArrayPtr. AbsAddr^ [
    AzCosProdArrayPtr. AbsAddr^ [ScanPtDispYAzCosProdFact] +
    AzSinProdArrayPtr. AbsAddr^ [ScanPtDispYAzSinProdFact]];

    ScanPtPosZ := ScannerPosZ + ScanPtDispArrayPtr. AbsAddr^ [
    AzCosProdArrayPtr. AbsAddr^ [ScanPtDispZAzCosProdFact] +
    AzSinProdArrayPtr. AbsAddr^ [ScanPtDispZAzSinProdFact]];

    WHILE ScanPtPosDatumBufferFullPtr. AbsAddr^ DO;

    ScanPtPosXPtr. AbsAddr^ := ScanPtPosX;
    ScanPtPosYPtr. AbsAddr^ := ScanPtPosY;
    ScanPtPosZPtr. AbsAddr^ := ScanPtPosZ;

    ScanPtPosDatumBufferFullPtr. AbsAddr^ := True;
    ScanPtPosDatumBufferFullUnPro3Ptr. AbsAddr^ := True

  END

END
ELSE
IF RangeDataConversionToBegin
THEN
RangeDataConversionInProcess := ScanPointNumber. IntVal = 15
ELSE
RangeDataConversionToBegin :=
(ScanLineNumber. IntVal = 15) AND (PrevScanLineNumber = 14);

IF ScanPointNumber. IntVal = 15
THEN
BEGIN
```

```
Brds12CommIdleBufferBusyFromBrd2Ptr. AbsAddr^ := True;

REPEAT
UNTIL NOT Brds12CommIdleBufferBusyFromBrd1Ptr. AbsAddr^;

IF Brds12CommIdleBufferPtrFromBrd2Ptr. AbsAddr^^. NewData
THEN
BEGIN

  Brds12CommTempBufferPtr :=
  Brds12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^;

  Brds12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^ :=
  Brds12CommIdleBufferPtrFromBrd2Ptr. AbsAddr^;

  Brds12CommIdleBufferPtrFromBrd2Ptr. AbsAddr^ :=
  Brds12CommTempBufferPtr;

  Brds12CommTempBufferPtr :=
  Brds12CommOtptBufferPtrFromBrd1Ptr. AbsAddr^;

  Brds12CommOtptBufferPtrFromBrd1Ptr. AbsAddr^ :=
  Brds12CommIdleBufferPtrFromBrd1Ptr. AbsAddr^;

  Brds12CommIdleBufferPtrFromBrd1Ptr. AbsAddr^ :=
  Brds12CommTempBufferPtr;

  Brds12CommIdleBufferPtrFromBrd2Ptr. AbsAddr^^. NewData := False

END;

Brds12CommIdleBufferBusyFromBrd2Ptr. AbsAddr^ := False;

ScanPtDispXElCosProdFact :=
Brds12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^^.
ScnnrToEarthTransMatrix. Rotat [X] [X];
ScanPtDispYElCosProdFact :=
Brds12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^^.
ScnnrToEarthTransMatrix. Rotat [X] [Y];
ScanPtDispZElCosProdFact :=
Brds12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^^.
ScnnrToEarthTransMatrix. Rotat [X] [Z];

ScanPtDispXAzSinProdFact :=
Brds12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^^.
ScnnrToEarthTransMatrix. Rotat [Y] [X];
ScanPtDispYAzSinProdFact :=
Brds12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^^.
ScnnrToEarthTransMatrix. Rotat [Y] [Y];
ScanPtDispZAzSinProdFact :=
Brds12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^^.
ScnnrToEarthTransMatrix. Rotat [Y] [Z];

ScanPtDispXElSinProdFact :=
Brds12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^^.
ScnnrToEarthTransMatrix. Rotat [Z] [X];
ScanPtDispYElSinProdFact :=
Brds12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^^.
ScnnrToEarthTransMatrix. Rotat [Z] [Y];
ScanPtDispZElSinProdFact :=
        Brds12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^^.
        ScnnrToEarthTransMatrix. Rotat [Z] [Z];

        ScannerPosX :=
        Brds12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^^.
        ScnnrToEarthTransMatrix. Trans [X];
        ScannerPosY :=
        Brds12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^^.
        ScnnrToEarthTransMatrix. Trans [Y];
        ScannerPosZ :=
        Brds12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^^.
        ScnnrToEarthTransMatrix. Trans [Z];

        NextScanLineNumber. IntVal := ScanLineNumber. IntVal + 1;
        IF NextScanLineNumber. IntVal = 16
        THEN
        NextScanLineNumber. IntVal := 0;

        ElCosProdArrayPtr. OffAddr. HiChar := NextScanLineNumber. LoChar;
        ElSinProdArrayPtr. OffAddr. HiChar := NextScanLineNumber. LoChar;

        ScanPtDispXAzCosProdFact :=
        ElCosProdArrayPtr. AbsAddr^ [ScanPtDispXElCosProdFact] +
        ElSinProdArrayPtr. AbsAddr^ [ScanPtDispXElSinProdFact];

        ScanPtDispYAzCosProdFact :=
        ElCosProdArrayPtr. AbsAddr^ [ScanPtDispYElCosProdFact] +
        ElSinProdArrayPtr. AbsAddr^ [ScanPtDispYElSinProdFact];

        ScanPtDispZAzCosProdFact :=
        ElCosProdArrayPtr. AbsAddr^ [ScanPtDispZElCosProdFact] +
        ElSinProdArrayPtr. AbsAddr^ [ScanPtDispZElSinProdFact];

        ScanPointNumber. IntVal := -1

     END

   END

END.
```

APPENDIX E

ELEVATION MAP STORAGE BOARD PROGRAM LISTING

```
MODULE TerrainElevationDataStorageProgram;

PROGRAM TerrainElevationDataStorageProgram (Input, Output);

  CONST

    maximumAllowableTerrainElevationDifference = 2;

  TYPE

    Bits = (Bit0, Bit1, Bit2, Bit3, Bit4, Bit5, Bit6, Bit7);
    SetOfBits = SET OF Bits;

    Byte =
    RECORD
      CASE Integer OF
        0: (BitSet: SetOfBits);
        1: (Character: Char)
    END;

    Word =
    RECORD
      CASE Integer OF
        0: (LowerOrderByte,
            HigherOrderByte: SetOfBits);
        1: (TwoBytes: Integer);
        2: (LowerOrderCharacter,
            HigherOrderCharacter: Char)
    END;

    BooleanPointer =
    RECORD
      CASE Integer OF
        0: (AbsoluteAddress: ^Boolean);
        1: (OffsetAddress,
            SegmentAddress: Integer)
    END;

    BytePointer =
    RECORD
      CASE Integer OF
        0: (AbsoluteAddress: ^Byte);
        1: (OffsetAddress,
            SegmentAddress: Integer)
    END;

    IntegerPointer =
    RECORD
      CASE Integer OF
        0: (AbsoluteAddress: ^Integer);
        1: (OffsetAddress,
            SegmentAddress: Integer)
    END;

  VAR

    ZeroByte: Byte;
    NoDatumWord: Word;
    MapXIndex,
    MapYIndex,
    MapXIndexTimes16,
    MapYIndexTimes16: Integer;
    TerrainX,
    TerrainY,
    TerrainZ: Integer;
    TerrainMapX,
    TerrainMapY: Integer;
    TerrainMapXIndex,
    TerrainMapYIndex: word;
    TerrainDatumBufferFullPointer,
    Board2TerrainDatumBufferFullPointer: BooleanPointer;
    TerrainXPointer,
    TerrainYPointer,
    TerrainZPointer: IntegerPointer;
    TerrainArrayXPositionPointer,
    TerrainArrayYPositionPointer,
    TerrainArrayPointer: IntegerPointer;


BEGIN

  Board2TerrainDatumBufferFullPointer. SegmentAddress := - 16385;
  Board2TerrainDatumBufferFullPointer. OffsetAddress  :=       0;

  Board2TerrainDatumBufferFullPointer. AbsoluteAddress^ := True;

  TerrainDatumBufferFullPointer. SegmentAddress := 8191;
  TerrainDatumBufferFullPointer. OffsetAddress  :=    0;

  TerrainDatumBufferFullPointer. AbsoluteAddress^ := True;

  TerrainXPointer. SegmentAddress := 8191;
  TerrainXPointer. OffsetAddress  :=    2;

  TerrainYPointer. SegmentAddress := 8191;
  TerrainYPointer. OffsetAddress  :=    4;

  TerrainZPointer. SegmentAddress := 8191;
  TerrainZPointer. OffsetAddress  :=    6;

  TerrainArrayXPositionPointer. SegmentAddress := 8192;
  TerrainArrayYPositionPointer. SegmentAddress := 8224;

  ZeroByte. Character := Chr (000H);

  NoDatumWord. HigherOrderCharacter := Chr (080H);
  NoDatumWord. LowerOrderCharacter  := Chr (000H);

  FOR MapXIndex := 0 TO 255 DO
  BEGIN

    TerrainArrayXPositionPointer. OffsetAddress := MapXIndex * 2;

    TerrainArrayXPositionPointer. AbsoluteAddress^ := NoDatumWord. TwoBytes

  END;

  FOR MapYIndex := 0 TO 255 DO
  BEGIN
  TerrainArrayYPositionPointer. OffsetAddress := MapYIndex * 2;

  TerrainArrayYPositionPointer. AbsoluteAddress^ := NoDatumWord. TwoBytes

END;

FOR MapYIndex := 0 TO 255 DO
FOR MapXIndex := 0 TO 255 DO
BEGIN

  TerrainArrayPointer. SegmentAddress := MapYIndex * 32 + 8256;
  TerrainArrayPointer. OffsetAddress  := MapXIndex *  2;

  TerrainArrayPointer. AbsoluteAddress^ := NoDatumWord. TwoBytes

END;

TerrainDatumBufferFullPointer. AbsoluteAddress^ := False;
Board2TerrainDatumBufferFullPointer. AbsoluteAddress^ := False;

WHILE True DO
BEGIN

  WHILE NOT TerrainDatumBufferFullPointer. AbsoluteAddress^ DO;

  TerrainX := TerrainXPointer. AbsoluteAddress^;
  TerrainY := TerrainYPointer. AbsoluteAddress^;
  TerrainZ := TerrainZPointer. AbsoluteAddress^;

  TerrainDatumBufferFullPointer. AbsoluteAddress^ := False;
  Board2TerrainDatumBufferFullPointer. AbsoluteAddress^ := False;

  TerrainMapX := TerrainX DIV 4;
  TerrainMapY := TerrainY DIV 4;

  TerrainMapXIndex      . TwoBytes := TerrainMapX;

  TerrainMapXIndex      . HigherOrderCharacter := ZeroByte. Character;

  TerrainMapYIndex      . TwoBytes := TerrainMapY;

  TerrainMapYIndex      . HigherOrderCharacter := ZeroByte. Character;

  TerrainArrayXPositionPointer. OffsetAddress :=
  TerrainMapXIndex. TwoBytes * 2;
  TerrainArrayYPositionPointer. OffsetAddress :=
  TerrainMapYIndex. TwoBytes * 2;

  IF TerrainArrayXPositionPointer. AbsoluteAddress^ <> TerrainMapX
  THEN
  BEGIN

    TerrainArrayPointer. OffsetAddress :=
    TerrainMapXIndex. TwoBytes * 2;

    MapYIndexTimes16 := -16;

    FOR MapYIndex := 0 TO 255 DO
    BEGIN

      MapYIndexTimes16 := MapYIndexTimes16 + 16;
```

```
        TerrainArrayPointer. SegmentAddress :=
        MapXIndexTimes16 * 2 +  8256;
        TerrainArrayPointer. AbsoluteAddress^ := NoDatumWord. TwoBytes

      END;

      TerrainArrayXPositionPointer. AbsoluteAddress^ := TerrainMapX

    END;

    IF TerrainArrayYPositionPointer. AbsoluteAddress^ <> TerrainMapY
    THEN
    BEGIN

      TerrainArrayPointer. SegmentAddress :=
      TerrainMapYIndex. TwoBytes * 32 + 8256;

      MapXIndexTimes16 := -16;

      FOR MapXIndex := 0 TO 255 DO
      BEGIN

        MapXIndexTimes16 := MapXIndexTimes16 + 16;

        TerrainArrayPointer. OffsetAddress :=
        MapXIndex * 2;
        TerrainArrayPointer. AbsoluteAddress^ := NoDatumWord. TwoBytes

      END;

      TerrainArrayYPositionPointer. AbsoluteAddress^ := TerrainMapY

    END;

    TerrainArrayPointer       . SegmentAddress :=
    TerrainMapYIndex. TwoBytes * 32 + 8256;
    TerrainArrayPointer       . OffsetAddress  :=
    TerrainMapXIndex. TwoBytes *  2;

    TerrainArrayPointer. AbsoluteAddress^ := TerrainZ

  END

END.
```

APPENDIX F

VEHICLE GUIDANCE BOARD PROGRAM LISTING

```
DegOfFrdm = RECORD
  Rotat: OrInEulerAngles;
  Trans: PtInCrd
END;

TransMatrix =
RECORD
  Rotat: ARRAY [Crd] OF VcInCrd;
  Trans: PtInCrd
END;

Legs = (None, FtLt, FtRt, CrLt, CrRt, RrLt, RrRt);

VehclBdyCmndType =
RECORD
  VehclToEarthTransMatrix: TransMatrix;
  LinVelInVehclCrd: VcInVehclCrd;
  AngVelInVehclCrd: VcInVehclCrd
END;

VehclBdyStt =
RECORD
  Time: Real;
  VehclToEarthTransMatrix: TransMatrix
END;

VehclBdyTrajType =
RECORD
  MaxBdySttIdx: Integer;
  VehclBdyStts: ARRAY [0..MaxBdySttsInBdyTraj] OF VehclBdyStt
END;

SptSttType = (Trnsfer, Support);

VehclLegStt =
RECORD
  SptStt: SptSttType;
  PosInEarthCrd: PtInEarthCrd
END;

VehclLegsSttsType = ARRAY [Legs] OF VehclLegStt;

VehclLegTrajType =
RECORD
  LftTime: Real;
  LftHgt: Real;
  PlcTime: Real;
  PlcPosInVehclCrd: PtInVehclCrd;
  CttTime: Real;
  CttHgtMin: Real;
  CttHgtMax: Real;
  NxtFndInEarthCrd: PtInEarthCrd
END;

VehclLegTrajsRec =
RECORD
  MaxLegTrajIdx: Integer;
  VehclLegTrajs: ARRAY [0..MaxLegTrajsInLegTrajsRec] OF VehclLegTrajType
END;

VehclLegsTrajsType = ARRAY [Legs] OF VehclLegTrajsRec;
```

```
MODULE VehicleTrajectoryPlanning;

PROGRAM VehicleTrajectoryPlanning (Input, Output);

  CONST

    Pi = 3.141592654;

    MaxBdySttsInBdyTraj = 60;
    MaxLegTrajsInLegTrajsRec = 10;

    VehclCtrToVehclTop     =   0.0;
    VehclCtrToVehclLtLegs =   1.625;
    VehclCtrToVehclRtLegs = - 1.625;
    VehclCtrToVehclFtLegs =   5.0;
    VehclCtrToVehclCrLegs =   0.0;
    VehclCtrToVehclRrLegs = - 5.0;
    VehclCtrToVehclCG      = - 2.333333;

    VehclAltitude = 6.0;

    VehclMaxTransVel = 8.0;
    VehclMaxTransAcc = 4.0;
    VehclMaxRotatVel = 0.5;
    VehclMaxRotatAcc = 0.25;
    VehclMinTurngRad = 16.0;

    MaxLegRtnTim = 0.8;
    LegRtnTim = 0.8;
    LegLftTim = 0.2;
    LegPlcTim = 0.2;

    GuidAlgrthmExecIntrvl = 0.25000;

  TYPE

    TwoCharInteger =
    RECORD
      CASE integer OF
        0: (LoChar,
            HiChar: Char);
        1: (ItgrVal: integer)
    END;

    Crd = (X, Y, Z);
    PtInCrd = ARRAY [Crd] OF Real;
    VcInCrd = ARRAY [Crd] OF Real;

    EarthCrd = (EarthX, EarthY, EarthZ);
    PtInEarthCrd = ARRAY [EarthCrd] OF Real;
    VcInEarthCrd = ARRAY [EarthCrd] OF Real;

    VehclCrd = (VehclX, VehclY, VehclZ);
    PtInVehclCrd = ARRAY [VehclCrd] OF Real;
    VcInVehclCrd = ARRAY [VehclCrd] OF Real;

    EulerAngles = (Yaw, Pch, Rll);
    OrInEulerAngles = ARRAY [EulerAngles] OF Real;
```

```
VehclLegCmnd =
RECORD
  SptStt: SptSttType;
  VehclLegCmndTraj: VehclLegTrajType
END;

VehclLegsCmndsType = ARRAY [Legs] OF VehclLegCmnd;

BooleanPtr =
RECORD
  CASE Integer OF
    0: (AbsAddr: ^Boolean);
    1: (OffAddr,
        SegAddr: Integer)
END;

IntegerPointer =
RECORD
  CASE Integer OF
    0: (AbsAddr: ^Integer);
    1: (OffAddr,
        SegAddr: Integer)
END;

Bros14CommBuffer =
RECORD
  CurrntTime: Real;
  VehclToEarthTransMatrix: TransMatrix;
  VehclLinVelInVehclCrd,
  VehclAngVelInVehclCrd: VcInVehclCrd;
  VehclLegsStts: VehclLegsSttsType;
  FrwdVelRqst,
  SideVelRqst,
  TurnVelRqst: Real;
  VehclLegsCmnds: VehclLegsCmndsType;
  NewData: Boolean
END;

Bros14CommBufferPtr =
RECORD
  CASE Integer OF
    0: (AbsAddr: ^Bros14CommBuffer);
    1: (OffAddr,
        SegAddr: Integer)
END;

Bros14CommBufferPtrPtr =
RECORD
  CASE Integer OF
    0: (AbsAddr: ^Bros14CommBufferPtr);
    1: (OffAddr,
        SegAddr: Integer)
END;

Bros41CommBuffer =
RECORD
  VehclBdyTraj: VehclBdyTrajType;
  VehclLegsTrajs: VehclLegsTrajsType;
  NewData: Boolean
END;
```

```
    Brds41CommBufferPtr =
    RECORD
      CASE Integer OF
        0: (AbsAddr: ^Brds41CommBuffer);
        1: (OffAddr,
            SegAddr: Integer)
    END;

    Brds41CommBufferPtrPtr =
    RECORD
      CASE Integer OF
        0: (AbsAddr: ^Brds41CommBufferPtr);
        1: (OffAddr,
            SegAddr: Integer)
    END;

  VAR

    Brd4InOperationPtr,
    Brd1InOperationPtr: BooleanPtr;
    Brds14CommInptBufferPtrFromBrd1Ptr,
    Brds14CommIdleBufferPtrFromBrd1Ptr,
    Brds14CommOtptBufferPtrFromBrd1Ptr,
    Brds14CommInptBufferPtrFromBrd4Ptr,
    Brds14CommIdleBufferPtrFromBrd4Ptr,
    Brds14CommOtptBufferPtrFromBrd4Ptr: Brds14CommBufferPtrPtr;
    Brds14CommTempBufferPtr: Brds14CommBufferPtr;
    Brds14CommIdleBufferBusyFromBrd1Ptr,
    Brds14CommIdleBufferBusyFromBrd4Ptr: BooleanPtr;
    Brds41CommInptBufferPtrFromBrd1Ptr,
    Brds41CommIdleBufferPtrFromBrd1Ptr,
    Brds41CommOtptBufferPtrFromBrd1Ptr,
    Brds41CommInptBufferPtrFromBrd4Ptr,
    Brds41CommIdleBufferPtrFromBrd4Ptr,
    Brds41CommOtptBufferPtrFromBrd4Ptr: Brds41CommBufferPtrPtr;
    Brds41CommTempBufferPtr: Brds41CommBufferPtr;
    Brds41CommIdleBufferBusyFromBrd1Ptr,
    Brds41CommIdleBufferBusyFromBrd4Ptr: BooleanPtr;
    TrrnArrXPosPtr,
    TrrnArrYPosPtr,
    TrrnArrXPosBelowPtr,
    TrrnArrXPosAbovePtr,
    TrrnArrYPosBelowPtr,
    TrrnArrYPosAbovePtr: IntegerPointer;
    NoDatItgr: TwoCharInteger;
    NoDatReal: Real;
    PlusInfinity: Real;
    VehclCGInVehclCrd: PtInVehclCrd;
    LegBasePosInVehclCrd: ARRAY [Legs] OF PtInVehclCrd;
    TransVelRqstMag: Real;
    TransAccRqstMag, RotatAccRqstMag: Real;
    VelRedFctr, AccRedFctr: Real;
    VehclBodyTraj: VehclBodyTrajType;
    VehclLegsTrajs: VehclLegsTrajsType;
    Leg: Legs;


  FUNCTION TrrnElev (
  TrrnX,
  TrrnY: Real) :
```

```
 Real;

   VAR

      TrrnMapX,
      TrrnMapY: Integer;
      TrrnMapXIdx,
      TrrnMapYIdx: TwoCharInteger;
      TrrnArrPtr: IntegerPointer;
      TrrnElevItgr: Integer;

 BEGIN

   TrrnMapX := Round (TrrnX * 8.0) DIV 4;

   TrrnMapY := Round (TrrnY * 8.0) DIV 4;

   TrrnMapXIdx. ItgrVal := TrrnMapX;
   TrrnMapXIdx. HiChar := Chr (000H);

   TrrnMapYIdx. ItgrVal := TrrnMapY;
   TrrnMapYIdx. HiChar := Chr (000H);

   TrrnArrXPosPtr. OffAddr := TrrnMapXIdx. ItgrVal * 2;
   TrrnArrYPosPtr. OffAddr := TrrnMapYIdx. ItgrVal * 2;

   IF
   (TrrnArrXPosPtr. AbsAddr^ <> TrrnMapX) OR
   (TrrnArrYPosPtr. AbsAddr^ <> TrrnMapY)
   THEN
   TrrnElev := NoDatReal
   ELSE
   BEGIN

     TrrnArrPtr. SegAddr := TrrnMapYIdx. ItgrVal * 32 + d256;
     TrrnArrPtr. OffAddr := TrrnMapXIdx. ItgrVal *  2;

     TrrnElevItgr := TrrnArrPtr. AbsAddr^;

     IF TrrnElevItgr = NoDatItgr. ItgrVal
     THEN
     TrrnElev := NoDatReal
     ELSE
     TrrnElev := TrrnElevItgr / 8.0

   END

END;


FUNCTION TrrnEthd (
TrrnX,
TrrnY: Real):
Real;

  VAR

     TrrnMapX,
     TrrnMapY,
     TrrnMapXBelow,
     TrrnMapXAbove,
    TrrnMapYBelow,
    TrrnMapYAbove: Integer;
    TrrnMapXIdx,
    TrrnMapYIdx,
    TrrnMapXIdxBelow,
    TrrnMapXIdxAbove,
    TrrnMapYIdxBelow,
    TrrnMapYIdxAbove: TwoCharInteger;
    TrrnArrPtr: IntegerPointer;
    TrrnElevItgr,
    AdjTrrnElevItgr: Integer;

BEGIN

  TrrnMapX := Round (TrrnX * 8.0) DIV 4;
  TrrnMapY := Round (TrrnY * 8.0) DIV 4;

  TrrnMapXBelow := TrrnMapX - 1;
  TrrnMapXAbove := TrrnMapX + 1;
  TrrnMapYBelow := TrrnMapY - 1;
  TrrnMapYAbove := TrrnMapY + 1;

  TrrnMapXIdx. ItgrVal := TrrnMapX;
  TrrnMapXIdx. HiChar  := Chr (0);
  TrrnMapYIdx. ItgrVal := TrrnMapY;
  TrrnMapYIdx. HiChar  := Chr (0);

  TrrnMapXIdxBelow. ItgrVal := TrrnMapXBelow;
  TrrnMapXIdxBelow. HiChar  := Chr (0);
  TrrnMapXIdxAbove. ItgrVal := TrrnMapXAbove;
  TrrnMapXIdxAbove. HiChar  := Chr (0);
  TrrnMapYIdxBelow. ItgrVal := TrrnMapYBelow;
  TrrnMapYIdxBelow. HiChar  := Chr (0);
  TrrnMapYIdxAbove. ItgrVal := TrrnMapYAbove;
  TrrnMapYIdxAbove. HiChar  := Chr (0);

  TrrnArrXPosPtr. OffAddr := TrrnMapXIdx. ItgrVal * 2;
  TrrnArrYPosPtr. OffAddr := TrrnMapYIdx. ItgrVal * 2;

  TrrnArrXPosBelowPtr. OffAddr := TrrnMapXIdxBelow. ItgrVal * 2;
  TrrnArrXPosAbovePtr. OffAddr := TrrnMapXIdxAbove. ItgrVal * 2;
  TrrnArrYPosBelowPtr. OffAddr := TrrnMapYIdxBelow. ItgrVal * 2;
  TrrnArrYPosAbovePtr. OffAddr := TrrnMapYIdxAbove. ItgrVal * 2;

  IF
  (TrrnArrXPosPtr. AbsAddr^ <> TrrnMapX) OR
  (TrrnArrYPosPtr. AbsAddr^ <> TrrnMapY) OR
  (TrrnArrXPosBelowPtr. AbsAddr^ <> TrrnMapXBelow) OR
  (TrrnArrXPosAbovePtr. AbsAddr^ <> TrrnMapXAbove) OR
  (TrrnArrYPosBelowPtr. AbsAddr^ <> TrrnMapYBelow) OR
  (TrrnArrYPosAbovePtr. AbsAddr^ <> TrrnMapYAbove)
  THEN
  TrrnEthq := NoDatReal
  ELSE
  BEGIN

    TrrnArrPtr. SegAddr := TrrnMapYIdx. ItgrVal * 32 + $256;
    TrrnArrPtr. OffAddr := TrrnMapXIdx. ItgrVal *  2;

    TrrnElevItgr := TrrnArrPtr. AbsAddr^;
```

```
IF TrrnElevItgr = NoDatItgr. ItgrVal
THEN
TrrnFtnd := NoDatReal
ELSE
BEGIN

  TrrnArrPtr. SegAddr := TrrnMapYIdx      . ItgrVal * 32 + 8256;
  TrrnArrPtr. OffAddr := TrrnMapXIdxBelow. ItgrVal *  2;

  AdjTrrnElevItgr := TrrnArrPtr. AbsAddr^;

  IF AdjTrrnElevItgr = NoDatItgr. ItgrVal
  THEN
  TrrnFtnd := NoDatReal
  ELSE
  IF Abs (TrrnElevItgr - AdjTrrnElevItgr) > 3
  THEN
  TrrnFtnd := NoDatReal
  ELSE
  BEGIN

    TrrnArrPtr. SegAddr := TrrnMapYIdx      . ItgrVal * 32 + 8256;
    TrrnArrPtr. OffAddr := TrrnMapXIdxAbove. ItgrVal *  2;

    AdjTrrnElevItgr := TrrnArrPtr. AbsAddr^;

    IF AdjTrrnElevItgr = NoDatItgr. ItgrVal
    THEN
    TrrnFtnd := NoDatReal
    ELSE
    IF Abs (TrrnElevItgr - AdjTrrnElevItgr) > 3
    THEN
    TrrnFtnd := NoDatReal
    ELSE
    BEGIN

      TrrnArrPtr. SegAddr := TrrnMapYIdxBelow. ItgrVal * 32 + 8256;
      TrrnArrPtr. OffAddr := TrrnMapXIdx      . ItgrVal *  2;

      AdjTrrnElevItgr := TrrnArrPtr. AbsAddr^;

      IF AdjTrrnElevItgr = NoDatItgr. ItgrVal
      THEN
      TrrnFtnd := NoDatReal
      ELSE
      IF Abs (TrrnElevItgr - AdjTrrnElevItgr) > 3
      THEN
      TrrnFtnd := NoDatReal
      ELSE
      BEGIN

        TrrnArrPtr. SegAddr := TrrnMapYIdxAbove. ItgrVal * 32 + 8256;
        TrrnArrPtr. OffAddr := TrrnMapXIdx      . ItgrVal *  2;

        AdjTrrnElevItgr := TrrnArrPtr. AbsAddr^;

        IF AdjTrrnElevItgr = NoDatItgr. ItgrVal
        THEN
        TrrnFtnd := NoDatReal
        ELSE
        IF Abs (TrrnElevItgr - AdjTrrnElevItgr) > 3
```

```
                THEN
                TrrnFtnd := NoDatReal
                ELSE
                TrrnFtnd := TrrnElevator / 8.0

              END
            END
          END
        END
      END
    END;


    FUNCTION ArcTan2 (
    Xvalue,
    Yvalue: Real):
    Real;

    BEGIN

      IF XValue > 0.0
      THEN
      ArcTan2 := ArcTan (YValue / XValue)
      ELSE
      IF XValue < 0.0
      THEN
      ArcTan2 := ArcTan (YValue / XValue) + Pi
      ELSE
      IF YValue > 0.0
      THEN
      ArcTan2 :=   Pi / 2.0
      ELSE
      ArcTan2 := - Pi / 2.0

    END;


    PROCEDURE CalcTransMatrixFromPosDegOfFrom (
    VAR TransMatrixVar: TransMatrix;
    DegOfFromVar: DegOfFrom);

      VAR

        CosYaw, SinYaw,
        CosPch, SinPch,
        CosRll, SinRll: Real;

    BEGIN

      CosYaw := Cos (DegOfFromVar. Rotat [Yaw]);
      SinYaw := Sin (DegOfFromVar. Rotat [Yaw]);
      CosPch := Cos (DegOfFromVar. Rotat [Pch]);
      SinPch := Sin (DegOfFromVar. Rotat [Pch]);
      CosRll := Cos (DegOfFromVar. Rotat [Rll]);
      SinRll := Sin (DegOfFromVar. Rotat [Rll]);

      TransMatrixVar. Rotat [X] [X] := CosYaw * CosPch;
      TransMatrixVar. Rotat [X] [Y] := SinYaw * CosPch;
      TransMatrixVar. Rotat [X] [Z] :=        - SinPch;

      TransMatrixVar. Rotat [Y] [X] :=
   CosYaw * SinPch * SinRll - SinYaw * CosRll;
   TransMatrixVar. Rotat [Y] [Y] :=
   SinYaw * SinPch * SinRll + CosYaw * CosRll;
   TransMatrixVar. Rotat [Y] [Z] :=
   (* *)    CosPch * SinRll;

   TransMatrixVar. Rotat [Z] [X] :=
   CosYaw * SinPch * CosRll + SinYaw * SinRll;
   TransMatrixVar. Rotat [Z] [Y] :=
   SinYaw * SinPch * CosRll - CosYaw * SinRll;
   TransMatrixVar. Rotat [Z] [Z] :=
   (* *)    CosPch * CosRll;

   TransMatrixVar. Trans [X] := DegOfFromVar. Trans [X];
   TransMatrixVar. Trans [Y] := DegOfFromVar. Trans [Y];
   TransMatrixVar. Trans [Z] := DegOfFromVar. Trans [Z]

END;


PROCEDURE CalcPosDegOfFrdmFromTransMatrix (
VAR DegOfFromVar: DegOfFrom;
TransMatrixVar: TransMatrix);

BEGIN

   DegOfFromVar. Rotat [Yaw] := ArcTan2 (
   TransMatrixVar. Rotat [X] [X], TransMatrixVar. Rotat [X] [Y]);

   DegOfFromVar. Rotat [Pch] := ArcTan2 (
   SqRt (
   Sqr (TransMatrixVar. Rotat [Y] [Z]) + Sqr (TransMatrixVar. Rotat [Z] [Z])),
   - TransMatrixVar. Rotat [X] [Z]);

   DegOfFromVar. Rotat [Rll] := ArcTan2 (
   TransMatrixVar. Rotat [Z] [Z], TransMatrixVar. Rotat [Y] [Z]);

   DegOfFromVar. Trans [X] := TransMatrixVar. Trans [X];
   DegOfFromVar. Trans [Y] := TransMatrixVar. Trans [Y];
   DegOfFromVar. Trans [Z] := TransMatrixVar. Trans [Z]

END;


PROCEDURE TrnsfrmPtToEarthCrdFrVenclCrd (
VAR PtInEarthCrdVar: PtInEarthCrd;
VenclToEarthTransMatrix: TransMatrix;
PtInVenclCrdVar: PtInVenclCrd);

BEGIN

   PtInEarthCrdVar [EarthX] :=
   VenclToEarthTransMatrix. Rotat [X] [X] * PtInVenclCrdVar [VenclX] +
   VenclToEarthTransMatrix. Rotat [Y] [X] * PtInVenclCrdVar [VenclY] +
   VenclToEarthTransMatrix. Rotat [Z] [X] * PtInVenclCrdVar [VenclZ] +
   VenclToEarthTransMatrix. Trans [X];

   PtInEarthCrdVar [EarthY] :=
   VenclToEarthTransMatrix. Rotat [X] [Y] * PtInVenclCrdVar [VenclX] +
   VenclToEarthTransMatrix. Rotat [Y] [Y] * PtInVenclCrdVar [VenclY] +
   VenclToEarthTransMatrix. Rotat [Z] [Y] * PtInVenclCrdVar [VenclZ] +
   VenclToEarthTransMatrix. Trans [Y];

   PtInEarthCrdVar [EarthZ] :=
   VenclToEarthTransMatrix. Rotat [X] [Z] * PtInVenclCrdVar [VenclX] +
   VenclToEarthTransMatrix. Rotat [Y] [Z] * PtInVenclCrdVar [VenclY] +
   VenclToEarthTransMatrix. Rotat [Z] [Z] * PtInVenclCrdVar [VenclZ] +
   VenclToEarthTransMatrix. Trans [Z]

END;


PROCEDURE TrnsfrmVcToEarthCrdFrVenclCrd (
VAR VcInEarthCrdVar: VcInEarthCrd;
VenclToEarthTransMatrix: TransMatrix;
VcInVenclCrdVar: VcInVenclCrd);

BEGIN

   VcInEarthCrdVar [EarthX] :=
   VenclToEarthTransMatrix. Rotat [X] [X] * VcInVenclCrdVar [VenclX] +
   VenclToEarthTransMatrix. Rotat [Y] [X] * VcInVenclCrdVar [VenclY] +
   VenclToEarthTransMatrix. Rotat [Z] [X] * VcInVenclCrdVar [VenclZ];

   VcInEarthCrdVar [EarthY] :=
   VenclToEarthTransMatrix. Rotat [X] [Y] * VcInVenclCrdVar [VenclX] +
   VenclToEarthTransMatrix. Rotat [Y] [Y] * VcInVenclCrdVar [VenclY] +
   VenclToEarthTransMatrix. Rotat [Z] [Y] * VcInVenclCrdVar [VenclZ];

   VcInEarthCrdVar [EarthZ] :=
   VenclToEarthTransMatrix. Rotat [X] [Z] * VcInVenclCrdVar [VenclX] +
   VenclToEarthTransMatrix. Rotat [Y] [Z] * VcInVenclCrdVar [VenclY] +
   VenclToEarthTransMatrix. Rotat [Z] [Z] * VcInVenclCrdVar [VenclZ];

END;


PROCEDURE TrnsfrmPtToVenclCrdFrEarthCrd (
VAR PtInVenclCrdVar: PtInVenclCrd;
VenclToEarthTransMatrix: TransMatrix;
PtInEarthCrdVar: PtInEarthCrd);

   VAR

     TmpPtInEarthCrdVar: PtInEarthCrd;

BEGIN

   TmpPtInEarthCrdVar [EarthX] :=
   PtInEarthCrdVar [EarthX] - VenclToEarthTransMatrix. Trans [X];
   TmpPtInEarthCrdVar [EarthY] :=
   PtInEarthCrdVar [EarthY] - VenclToEarthTransMatrix. Trans [Y];
   TmpPtInEarthCrdVar [EarthZ] :=
   PtInEarthCrdVar [EarthZ] - VenclToEarthTransMatrix. Trans [Z];

   PtInVenclCrdVar [VenclX] :=
   VenclToEarthTransMatrix. Rotat [X] [X] * TmpPtInEarthCrdVar [EarthX] +
   VenclToEarthTransMatrix. Rotat [X] [Y] * TmpPtInEarthCrdVar [EarthY] +
   VenclToEarthTransMatrix. Rotat [X] [Z] * TmpPtInEarthCrdVar [EarthZ];

   PtInVenclCrdVar [VenclY] :=
   VenclToEarthTransMatrix. Rotat [Y] [X] * TmpPtInEarthCrdVar [EarthX] +
   VehclToEarthTransMatrix. Rotat [Y] [Y] * TmpPtInEarthCrdVar [EarthY] +
   VehclToEarthTransMatrix. Rotat [Y] [Z] * TmpPtInEarthCrdVar [EarthZ];

   PtInVehclCrdVar [VehclZ] :=
   VehclToEarthTransMatrix. Rotat [Z] [X] * TmpPtInEarthCrdVar [EarthX] +
   VehclToEarthTransMatrix. Rotat [Z] [Y] * TmpPtInEarthCrdVar [EarthY] +
   VehclToEarthTransMatrix. Rotat [Z] [Z] * TmpPtInEarthCrdVar [EarthZ]

END;


FUNCTION FtInLimits (
Leg: Legs;
FtPosInEarthCrd: PtInEarthCrd;
VehclToEarthTransMatrix: TransMatrix):
Boolean;

  CONST

    MaxLegLngth  =   7.833333;
    MinLegLngth  =   3.833333;
    MinAbdAngle  = - 0.085;
    MaxAbdAngle  =   0.430;
    MaxFwdDsplc  =   3.0;
    MinFwdDsplc  = - 3.0;

  VAR

    FtPosInVehclCrd: PtInVehclCrd;
    FtFwdDsplc,
    FtAbdAngle,
    FtLegLngth: Real;
    FtInFwdDsplcLimits,
    FtInAbdAngleLimits,
    FtInLegLngthLimits: Boolean;

BEGIN

  TrnsfrmPtToVehclCrdFrEarthCrd (
  FtPosInVehclCrd, VehclToEarthTransMatrix, FtPosInEarthCrd);

  FtFwdDsplc :=
  FtPosInVehclCrd [VehclX] - LegBasePosInVehclCrd [Leg] [VehclX];

  FtInFwdDsplcLimits :=
  (FtFwdDsplc >= MinFwdDsplc) AND (FtFwdDsplc <= MaxFwdDsplc);

  IF Odd (Ord (Leg))
  THEN
  FtAbdAngle := ArcTan2 (
  - FtPosInVehclCrd [VehclZ],
  (FtPosInVehclCrd [VehclY] - LegBasePosInVehclCrd [Leg] [VehclY]))
  ELSE
  FtAbdAngle := ArcTan2 (
  - FtPosInVehclCrd [VehclZ],
  - (FtPosInVehclCrd [VehclY] - LegBasePosInVehclCrd [Leg] [VehclY]));

  FtInAbdAngleLimits :=
  (FtAbdAngle >= MinAbdAngle) AND (FtAbdAngle <= MaxAbdAngle);

  FtLegLngth := SqRt (
  Sqr (FtPosInVehclCrd [VehclY] - LegBasePosInVehclCrd [Leg] [VehclY]) +
  Sqr (FtPosInVehclCrd [VehclZ] - LegBasePosInVehclCrd [Leg] [VehclZ]));

  FtInLegLngthLimits :=
  (FtLegLngth >= MinLegLngth) AND (FtLegLngth <= MaxLegLngth);

  FtInLimits :=
  FtInFwdDsplcLimits AND FtInAbdAngleLimits AND FtInLegLngthLimits

END;


FUNCTION CalcVehclBdyTraj (
FrwdAccRqst,
SideAccRqst,
TurnAccRqst: Real;
CurrntTime: Real;
VehclToEarthTransMatrix: TransMatrix;
VehclLinVelInVehclCrd,
VehclAngVelInVehclCrd: VcInVehclCrd;
VAR VehclBdyTraj: VehclBdyTrajType):
Boolean;

  VAR

    VehclPosDegOfFrdmWRtEarthCrd: DegOfFrdm;
    VehclEarthX, VehclEarthY, VehclYaw: Real;
    FrwdVel, SideVel, TurnVel: Real;
    NewVehclEarthX, NewVehclEarthY, NewVehclYaw: Real;
    NewFrwdVel, NewSideVel, NewTurnVel: Real;
    TimeIncrmnt: Real;
    FrwdDecRqst, SideDecRqst, TurnDecRqst: Real;
    DecTime: Real;


  PROCEDURE CalcNewVehclPos (
  VehclEarthX, VehclEarthY, VehclYaw: Real;
  FrwdVel, SideVel, TurnVel: Real;
  FrwdAccRqst, SideAccRqst, TurnAccRqst: Real;
  VAR NewVehclEarthX, NewVehclEarthY, NewVehclYaw: Real;
  VAR NewFrwdVel, NewSideVel, NewTurnVel: Real;
  VAR TimeIncrmnt: Real);

    CONST

      TransIncrmnt = 0.25;
      RotatIncrmnt = 0.015625;

    VAR

      FrwdTime, SideTime, TurnTime: Real;
      AveFrwdVel, AveSideVel, AveTurnVel: Real;
      AveTransVel, AveVehclYaw: Real;
      TurningRad, TurningArc: Real;
      TransVelAngWRtEarthCrd: Real;
      TurningCtrEarthX, TurningCtrEarthY: Real;


    FUNCTION MinDistTime (
    InitialVelocity,
    Acceleration,
Distance: Real):
Real;

  VAR

    B2mns4AC, B2pls4AC: Real;
    DistTime: Real;


BEGIN

  IF Acceleration = 0.0
  THEN
  IF InitialVelocity = 0.0
  THEN
  MinDistTime := PlusInfinity
  ELSE
  MinDistTime := Abs (Distance / InitialVelocity)
  ELSE
  BEGIN

    MinDistTime := PlusInfinity;

    B2mns4AC := Sqr (InitialVelocity) + 2.0 * Acceleration * Distance;

    IF B2mns4AC >= 0.0
    THEN
    BEGIN

      DistTime := - InitialVelocity + SqRt (B2mns4AC) / Acceleration;

      IF (DistTime >= 0.0) AND (DistTime < MinDistTime)
      THEN
      MinDistTime := DistTime;

      DistTime := - InitialVelocity - SqRt (B2mns4AC) / Acceleration;

      IF (DistTime >= 0.0) AND (DistTime < MinDistTime)
      THEN
      MinDistTime := DistTime

    END;

    B2pls4AC := Sqr (InitialVelocity) - 2.0 * Acceleration * Distance;

    IF B2pls4AC >= 0.0
    THEN
    BEGIN

      DistTime := - InitialVelocity + SqRt (B2pls4AC) / Acceleration;

      IF (DistTime >= 0.0) AND (DistTime < MinDistTime)
      THEN
      MinDistTime := DistTime;

      DistTime := - InitialVelocity - SqRt (B2pls4AC) / Acceleration;

      IF (DistTime >= 0.0) AND (DistTime < MinDistTime)
      THEN
      MinDistTime := DistTime
      END
    END
  END;

BEGIN

  FrwdTime := MinDistTime (FrwdVel, FrwdAccRqst, TransIncrmnt);
  SideTime := MinDistTime (SideVel, SideAccRqst, TransIncrmnt);
  TurnTime := MinDistTime (TurnVel, TurnAccRqst, RotatIncrmnt);

  IF (FrwdTime <= SideTime) AND (FrwdTime <= TurnTime)
  THEN
  TimeIncrmnt := FrwdTime
  ELSE
  IF (SideTime <= TurnTime)
  THEN
  TimeIncrmnt := SideTime
  ELSE
  TimeIncrmnt := TurnTime;

  NewFrwdVel := FrwdVel + FrwdAccRqst * TimeIncrmnt;
  NewSideVel := SideVel + SideAccRqst * TimeIncrmnt;
  NewTurnVel := TurnVel + TurnAccRqst * TimeIncrmnt;

  AveFrwdVel := (FrwdVel + NewFrwdVel) / 2.0;
  AveSideVel := (SideVel + NewSideVel) / 2.0;
  AveTurnVel := (TurnVel + NewTurnVel) / 2.0;

  NewVenclYaw := VenclYaw + AveTurnVel * TimeIncrmnt;

  IF (AveFrwdVel = 0.0) AND (AveSideVel = 0.0)
  THEN
  BEGIN

    NewVenclEarthX := VenclEarthX;
    NewVenclEarthY := VenclEarthY

  END
  ELSE
  IF AveTurnVel = 0.0
  THEN
  BEGIN

    NewVenclEarthX := VenclEarthX +
    AveFrwdVel * Cos (VenclYaw) - AveSideVel * Sin (VenclYaw);
    NewVenclEarthY := VenclEarthY +
    AveFrwdVel * Sin (VenclYaw) + AveSideVel * Cos (VenclYaw)

  END
  ELSE
  BEGIN

    AveTransVel := SqRt (Sqr (AveFrwdVel) + Sqr (AveSideVel));

    TurningRad := Abs (AveTransVel / AveTurnVel);

    AveVenclYaw := (VenclYaw + NewVenclYaw) / 2.0;

    TransVelAngWRTEarthCrd := AveVenclYaw +
    ArcTan2 (AveFrwdVel, AveSideVel);
    IF AveTurnVel > 0.0
    THEN
    BEGIN

      TurningCtrEarthX := VehclEarthX +
      TurningRad * Cos (TransVelAngWRTEarthCrd + Pi / 2.0);
      TurningCtrEarthY := VehclEarthY +
      TurningRad * Sin (TransVelAngWRTEarthCrd + Pi / 2.0)

    END
    ELSE
    BEGIN

      TurningCtrEarthX := VehclEarthX +
      TurningRad * Cos (TransVelAngWRTEarthCrd - Pi / 2.0);
      TurningCtrEarthY := VehclEarthY +
      TurningRad * Sin (TransVelAngWRTEarthCrd - Pi / 2.0)

    END;

    TurningArc := AveTransVel * TimeIncrmnt / TurningRad;

    IF AveTurnVel > 0.0
    THEN
    BEGIN

      NewVehclEarthX := TurningCtrEarthX +
      TurningRad * Cos (TransVelAngWRTEarthCrd - Pi / 2.0 + TurningArc);
      NewVehclEarthY := TurningCtrEarthY +
      TurningRad * Sin (TransVelAngWRTEarthCrd - Pi / 2.0 + TurningArc)

    END
    ELSE
    BEGIN

      NewVehclEarthX := TurningCtrEarthX +
      TurningRad * Cos (TransVelAngWRTEarthCrd + Pi / 2.0 - TurningArc);
      NewVehclEarthY := TurningCtrEarthY +
      TurningRad * Sin (TransVelAngWRTEarthCrd + Pi / 2.0 - TurningArc)

    END
  END
END;


FUNCTION CalcVehclToEarthTransmatrix (
VehclEarthX,
VehclEarthY,
VehclYaw: Real;
VAR VehclToEarthTransmatrix: Transmatrix):
Boolean;

  VAR

    VehclXIdx, VehclYIdx: Integer;
    TrrnPtsInEarthCrd: ARRAY [-2..2, -1..1] OF PtInEarthCrd;
    NumberTrrnPts: Integer;
    SumEarthX, SumEarthY, SumEarthZ: Real;
    MeanEarthX, MeanEarthY, MeanEarthZ: Real;
    SumEarthXEarthX,
    SumEarthXEarthY,
    SumEarthXEarthZ,
    SumEarthYEarthY,
    SumEarthYEarthZ: Real;
    ZIntercept, EarthXCoeff, EarthYCoeff: Real;
    XVcPtInEarthCrd: PtInEarthCrd;
    XVcPtMag: Real;


BEGIN

  FOR VehclYIdx := -1 TO 1 DO
  FOR VehclXIdx := -2 TO 2 DO
  BEGIN

    TrrnPtsInEarthCrd [VehclXIdx, VehclYIdx] [EarthX] := VehclEarthX +
    Cos (VehclYaw) * 4.0 * VehclXIdx - Sin (VehclYaw) * 4.0 * VehclYIdx;
    TrrnPtsInEarthCrd [VehclXIdx, VehclYIdx] [EarthY] := VehclEarthY +
    Sin (VehclYaw) * 4.0 * VehclYIdx + Cos (VehclYaw) * 4.0 * VehclYIdx;

    TrrnPtsInEarthCrd [VehclXIdx, VehclYIdx] [EarthZ] := TrrnFtnd (
    TrrnPtsInEarthCrd [VehclXIdx, VehclYIdx] [EarthX],
    TrrnPtsInEarthCrd [VehclXIdx, VehclYIdx] [EarthY]);

  END;

  NmbrTrrnPts := 0;
  SumEarthX := 0.0;
  SumEarthY := 0.0;
  SumEarthZ := 0.0;

  FOR VehclYIdx := -1 TO 1 DO
  FOR VehclXIdx := -2 TO 2 DO
  IF TrrnPtsInEarthCrd [VehclXIdx, VehclYIdx] [EarthZ] <> NoDatReal
  THEN
  BEGIN

    NmbrTrrnPts := NmbrTrrnPts + 1;

    SumEarthX := SumEarthX +
    TrrnPtsInEarthCrd [VehclXIdx, VehclYIdx] [EarthX];
    SumEarthY := SumEarthY +
    TrrnPtsInEarthCrd [VehclXIdx, VehclYIdx] [EarthY];
    SumEarthZ := SumEarthZ +
    TrrnPtsInEarthCrd [VehclXIdx, VehclYIdx] [EarthZ]

  END;

  IF NmbrTrrnPts >= 3
  THEN
  BEGIN

    MeanEarthZ := SumEarthZ / NmbrTrrnPts;

    FOR VehclYIdx := -1 TO 1 DO
    FOR VehclXIdx := -2 TO 2 DO
    IF
    (TrrnPtsInEarthCrd [VehclXIdx, VehclYIdx] [EarthZ] - MeanEarthZ ) > 2.0
    THEN
    BEGIN
```

```
      NmbrTrrnPts := NmbrTrrnPts - 1;

      SumEarthX := SumEarthX -
      TrrnPtsInEarthCrd [VenclXIdx, VenclYIdx] [EarthX];
      SumEarthY := SumEarthY -
      TrrnPtsInEarthCrd [VenclXIdx, VenclYIdx] [EarthY];
      SumEarthZ := SumEarthZ -
      TrrnPtsInEarthCrd [VenclXIdx, VenclYIdx] [EarthZ];

      TrrnPtsInEarthCrd [VenclXIdx, VenclYIdx] [EarthZ] := NoDatReal

    END
  END;

  CalcVenclToEarthTransMatrix := (NmbrTrrnPts >= 3);

  IF CalcVenclToEarthTransMatrix
  THEN
  BEGIN

    MeanEarthX := SumEarthX / NmbrTrrnPts;
    MeanEarthY := SumEarthY / NmbrTrrnPts;
    MeanEarthZ := SumEarthZ / NmbrTrrnPts;

    SumEarthXEarthX := 0.0;
    SumEarthXEarthY := 0.0;
    SumEarthXEarthZ := 0.0;
    SumEarthYEarthY := 0.0;
    SumEarthYEarthZ := 0.0;

    FOR VenclYIdx := -1 TO 1 DO
    FOR VenclXIdx := -2 TO 2 DO
    IF TrrnPtsInEarthCrd [VenclXIdx, VenclYIdx] [EarthZ] <> NoDatReal
    THEN
    BEGIN

      SumEarthXEarthX := SumEarthXEarthX +
      (TrrnPtsInEarthCrd [VenclXIdx, VenclYIdx] [EarthX] - MeanEarthX) *
      (TrrnPtsInEarthCrd [VenclXIdx, VenclYIdx] [EarthX] - MeanEarthX);
      SumEarthXEarthY := SumEarthXEarthY +
      (TrrnPtsInEarthCrd [VenclXIdx, VenclYIdx] [EarthX] - MeanEarthX) *
      (TrrnPtsInEarthCrd [VenclXIdx, VenclYIdx] [EarthY] - MeanEarthY);
      SumEarthXEarthZ := SumEarthXEarthZ +
      (TrrnPtsInEarthCrd [VenclXIdx, VenclYIdx] [EarthX] - MeanEarthX) *
      (TrrnPtsInEarthCrd [VenclXIdx, VenclYIdx] [EarthZ] - MeanEarthZ);
      SumEarthYEarthY := SumEarthYEarthY +
      (TrrnPtsInEarthCrd [VenclXIdx, VenclYIdx] [EarthY] - MeanEarthY) *
      (TrrnPtsInEarthCrd [VenclXIdx, VenclYIdx] [EarthY] - MeanEarthY);
      SumEarthYEarthZ := SumEarthYEarthZ +
      (TrrnPtsInEarthCrd [VenclXIdx, VenclYIdx] [EarthY] - MeanEarthY) *
      (TrrnPtsInEarthCrd [VenclXIdx, VenclYIdx] [EarthZ] - MeanEarthZ)

    END;

    CalcVenclToEarthTransMatrix := (SumEarthXEarthX <> 0.0);

    IF CalcVenclToEarthTransMatrix
    THEN
    BEGIN

      CalcVenclToEarthTransMatrix := ((
SumEarthXEarthY * SumEarthXEarthY -
SumEarthXEarthX * SumEarthYEarthY) <> 0.0);

IF CalcVenclToEarthTransMatrix
THEN
BEGIN

  EarthYCoeff := (
  SumEarthXEarthY * SumEarthXEarthZ -
  SumEarthXEarthX * SumEarthYEarthZ) / (
  SumEarthXEarthY * SumEarthXEarthY -
  SumEarthXEarthX * SumEarthYEarthY);

  EarthXCoeff := (SumEarthXEarthZ - EarthYCoeff * SumEarthXEarthY) /
  SumEarthXEarthX;

  ZIntercept := MeanEarthZ -
  EarthXCoeff * MeanEarthX - EarthYCoeff * MeanEarthY;

  XVcPtInEarthCrd [EarthX] := MeanEarthX + 10.0 * Cos (VenclYaw);
  XVcPtInEarthCrd [EarthY] := MeanEarthY + 10.0 * Sin (VenclYaw);
  XVcPtInEarthCrd [EarthZ] := ZIntercept +
  EarthXCoeff * XVcPtInEarthCrd [EarthX] +
  EarthYCoeff * XVcPtInEarthCrd [EarthY];

  XVcPtMag := SqRt (
  Sqr (XVcPtInEarthCrd [EarthX] - MeanEarthX) +
  Sqr (XVcPtInEarthCrd [EarthY] - MeanEarthY) +
  Sqr (0.5 * (XVcPtInEarthCrd [EarthZ] - MeanEarthZ)));

  VenclToEarthTransMatrix. Rotat [X] [X] :=
  (XVcPtInEarthCrd [EarthX] - MeanEarthX) / XVcPtMag;
  VenclToEarthTransMatrix. Rotat [X] [Y] :=
  (XVcPtInEarthCrd [EarthY] - MeanEarthY) / XVcPtMag;
  VenclToEarthTransMatrix. Rotat [X] [Z] := 0.5 *
  (XVcPtInEarthCrd [EarthZ] - MeanEarthZ) / XVcPtMag;

  VenclToEarthTransMatrix. Rotat [Y] [X] := - Sin (VenclYaw);
  VenclToEarthTransMatrix. Rotat [Y] [Y] :=   Cos (VenclYaw);
  VenclToEarthTransMatrix. Rotat [Y] [Z] := 0.0;

  VenclToEarthTransMatrix. Rotat [Z] [X] :=
  VenclToEarthTransMatrix. Rotat [X] [Y] *
  VenclToEarthTransMatrix. Rotat [Y] [Z] -
  VenclToEarthTransMatrix. Rotat [X] [Z] *
  VenclToEarthTransMatrix. Rotat [Y] [Y];
  VenclToEarthTransMatrix. Rotat [Z] [Y] :=
  VenclToEarthTransMatrix. Rotat [X] [Z] *
  VenclToEarthTransMatrix. Rotat [Y] [X] -
  VenclToEarthTransMatrix. Rotat [X] [X] *
  VenclToEarthTransMatrix. Rotat [Y] [Z];
  VenclToEarthTransMatrix. Rotat [Z] [Z] :=
  VenclToEarthTransMatrix. Rotat [X] [X] *
  VenclToEarthTransMatrix. Rotat [Y] [Y] -
  VenclToEarthTransMatrix. Rotat [X] [Y] *
  VenclToEarthTransMatrix. Rotat [Y] [X];

  VenclToEarthTransMatrix. Trans [X] := VenclEarthX;
  VenclToEarthTransMatrix. Trans [Y] := VenclEarthY;
  VenclToEarthTransMatrix. Trans [Z] := VenclAltitude + ZIntercept +
  EarthXCoeff * VenclEarthX + EarthYCoeff * VenclEarthY
```

```
        END
      END
    END
  END;


BEGIN

  CalcPosDegOfFrdmFromTransMatrix (
  VenclPosDegOfFrdmWRTEarthCrd, VenclToEarthTransMatrix);

  VenclEarthX := VenclPosDegOfFrdmWRTEarthCrd. Trans [X];
  VenclEarthY := VenclPosDegOfFrdmWRTEarthCrd. Trans [Y];
  VenclYaw    := VenclPosDegOfFrdmWRTEarthCrd. Rotat [Yaw];

  FrwdVel := VenclLinVelInVenclCrd [VenclX];
  SideVel := VenclLinVelInVenclCrd [VenclY];
  TurnVel := VenclAngVelInVenclCrd [VenclZ];

  VenclBdyTraj. MaxBdySttIdx := 0;

  VenclBdyTraj. VenclBdyStts [VenclBdyTraj. MaxBdySttIdx]. Time := CurrntTime;
  VenclBdyTraj. VenclBdyStts [VenclBdyTraj. MaxBdySttIdx].
  VenclToEarthTransMatrix := VenclToEarthTransMatrix;

  CalcVenclBdyTraj := True;

  IF
  (FrwdVel <> 0.0) OR (SideVel <> 0.0) OR (TurnVel <> 0.0) OR
  (FrwdAccRqst <> 0.0) OR (SideAccRqst <> 0.0) OR (TurnAccRqst <> 0.0)
  THEN
  BEGIN

    REPEAT

      VenclBdyTraj. MaxBdySttIdx := VenclBdyTraj. MaxBdySttIdx + 1;

      CalcNewVenclPos (
      VenclEarthX, VenclEarthY, VenclYaw,
      FrwdVel, SideVel, TurnVel,
      FrwdAccRqst, SideAccRqst, TurnAccRqst,
      NewVenclEarthX, NewVenclEarthY, NewVenclYaw,
      NewFrwdVel, NewSideVel, NewTurnVel,
      TimeIncrmnt);

      VenclBdyTraj. VenclBdyStts [VenclBdyTraj. MaxBdySttIdx]. Time :=
      VenclBdyTraj. VenclBdyStts [VenclBdyTraj. MaxBdySttIdx - 1]. Time +
      TimeIncrmnt;

      CalcVenclBdyTraj := CalcVenclToEarthTransMatrix (
      NewVenclEarthX, NewVenclEarthY, NewVenclYaw,
      VenclBdyTraj. VenclBdyStts [VenclBdyTraj. MaxBdySttIdx].
      VenclToEarthTransMatrix);

      FrwdVel := NewFrwdVel;
      SideVel := NewSideVel;
      TurnVel := NewTurnVel;

      VenclEarthX := NewVenclEarthX;
      VenclEarthY := NewVenclEarthY;
  VenclYaw    := newVenclYaw

UNTIL (NOT CalcVenclBdyTraj) OR ((VenclBdyTraj.
VenclBdyStts [VenclBdyTraj. maxBdySttIdx]. Time - CurrntTime) >
GuidAlgrthmExecIntrvl);

IF CalcVenclBdyTraj
THEN
IF (FrwdVel <> 0.0) OR (SideVel <> 0.0) OR (TurnVel <> 0.0)
THEN
BEGIN

  IF
  (Abs (FrwdVel) >= Abs (SideVel)) AND
  ((Abs (FrwdVel) / VenclMaxTransAcc) >=
  (Abs (TurnVel) / VenclMaxRotatAcc))
  THEN
  BEGIN

    FrwdDecRqst := - VenclMaxTransAcc * FrwdVel / Abs (FrwdVel);
    SideDecRqst := - VenclMaxTransAcc * SideVel / Abs (FrwdVel);
    TurnDecRqst := - VenclMaxRotatAcc *
    (TurnVel / VenclMaxRotatAcc) / (Abs (FrwdVel) / VenclMaxTransAcc)

  END
  ELSE
  IF
  ((Abs (SideVel) / VenclMaxTransAcc) >=
  (Abs (TurnVel) / VenclMaxRotatAcc))
  THEN
  BEGIN

    FrwdDecRqst := - VenclMaxTransAcc * FrwdVel / Abs (SideVel);
    SideDecRqst := - VenclMaxTransAcc * SideVel / Abs (SideVel);
    TurnDecRqst := - VenclMaxRotatAcc *
    (TurnVel / VenclMaxRotatAcc) / (Abs (SideVel) / VenclMaxTransAcc)

  END
  ELSE
  BEGIN

    FrwdDecRqst := - VenclMaxTransAcc *
    (FrwdVel / VenclMaxTransAcc) / (Abs (TurnVel) / VenclMaxRotatAcc);
    SideDecRqst := - VenclMaxTransAcc *
    (SideVel / VenclMaxTransAcc) / (Abs (TurnVel) / VenclMaxRotatAcc);
    TurnDecRqst := - VenclMaxRotatAcc * TurnVel / Abs (TurnVel)

  END;

  IF FrwdVel <> 0.0
  THEN
  DecTime := Abs (FrwdVel / FrwdDecRqst)
  ELSE
  IF SideVel <> 0.0
  THEN
  DecTime := Abs (SideVel / SideDecRqst)
  ELSE
  DecTime := Abs (TurnVel / TurnDecRqst);

  REPEAT
```

```
        VehclBdyTraj. MaxBdySttIdx := VehclBdyTraj. MaxBdySttIdx + 1;

        CalcNewVehclPos (
        VehclEarthX, VehclEarthY, VehclYaw,
        FrwdVel, SideVel, TurnVel,
        FrwdDecRqst, SideDecRqst, TurnDecRqst,
        NewVehclEarthX, NewVehclEarthY, NewVehclYaw,
        NewFrwdVel, NewSideVel, NewTurnVel,
        TimeIncrmnt);

        VehclBdyTraj. VehclBdyStts [VehclBdyTraj. MaxBdySttIdx]. Time :=
        VehclBdyTraj. VehclBdyStts [VehclBdyTraj. MaxBdySttIdx - 1]. Time +
        TimeIncrmnt;

        CalcVehclBdyTraj := CalcVehclToEarthTransMatrix (
        NewVehclEarthX, NewVehclEarthY, NewVehclYaw,
        VehclBdyTraj.
        VehclBdyStts [VehclBdyTraj. MaxBdySttIdx]. VehclToEarthTransMatrix);

        FrwdVel := NewFrwdVel;
        SideVel := NewSideVel;
        TurnVel := NewTurnVel;

        VehclEarthX := NewVehclEarthX;
        VehclEarthY := NewVehclEarthY;
        VehclYaw    := NewVehclYaw

      UNTIL (NOT CalcVehclBdyTraj) OR ((VehclBdyTraj.
      VehclBdyStts [VehclBdyTraj. MaxBdySttIdx]. Time - CurrntTime) >
      (GuidAlgrthmExecIntrvl + DecTime))

    END
  END
END;


FUNCTION CalcVehclLegsTrajs (
VehclBdyTraj: VehclBdyTrajType;
VehclLegsStts: VehclLegsSttsType;
VehclLegsCmnds: VehclLegsCmndsType;
VAR VehclLegsTrajs: VehclLegsTrajsType):
Boolean;

  VAR

    FutureVehclLegsStts: VehclLegsSttsType;
    LegTrajIdx: ARRAY [Legs] OF Integer;
    LegsOutOfLimitsIdxs: ARRAY [Legs] OF Integer;
    FutureTime: Real;
    PotentialVehclLegsStts: VehclLegsSttsType;
    LegsCanBeLifted: Boolean;
    MinLegOutOfLimitsIdx: Integer;
    LegMinLegOutOfLimits: Legs;
    VehclLegTraj: VehclLegTrajType;
    Leg: Legs;
    VehclCrdIdx: VehclCrd;
    EarthCrdIdx: EarthCrd;
    BdySttIdx: Integer;


  FUNCTION VehclIsStable (
VenclToEarthTransMatrix: TransMatrix;
VenclLegsStts: VenclLegsSttsType):
Boolean;

 CONST

   MinStabilityMargin = 0.5;

 VAR

   VehclCGInEarthCrd: PtInEarthCrd;
   FtToCGDstnc, FtToCGAngle: Real;
   FtToCGStabMrgnArc: Real;
   FtToCGAnglePls, FtToCGAngleMns: Real;
   FtToCGAnglePlsTst, FtToCGAngleMnsTst: Boolean;
   TstLineSlope: Real;
   Leg, OtherLeg: Legs;


BEGIN

 TrnsfrmPtToEarthCrdFrVenclCrd (
 VenclCGInEarthCrd, VenclToEarthTransMatrix, VehclCGInVehclCrd);

 VenclIsStable := True;

 FOR Leg := FtLt TO RrRt DO
 IF VenclIsStable
 THEN
 IF VenclLegsStts [Leg]. SptStt = Support
 THEN
 BEGIN

   FtToCGDstnc := SqRt (
   Sqr (
   VehclCGInEarthCrd [EarthX] -
   VenclLegsStts [Leg]. PosInEarthCrd [EarthX]) +
   Sqr (
   VenclCGInEarthCrd [EarthY] -
   VenclLegsStts [Leg]. PosInEarthCrd [EarthY]));

   IF FtToCGDstnc >= MinStabilityMargin
   THEN
   BEGIN

     FtToCGAngle := ArcTan2 (
     VehclCGInEarthCrd [EarthX] -
     VenclLegsStts [Leg]. PosInEarthCrd [EarthX],
     VenclCGInEarthCrd [EarthY] -
     VenclLegsStts [Leg]. PosInEarthCrd [EarthY]);

     FtToCGStabMrgnArc := ArcSin (MinStabilityMargin / FtToCGDstnc);

     FtToCGAnglePls := FtToCGAngle + FtToCGStabMrgnArc;
     IF FtToCGAnglePls >= 3.0 * Pi / 2.0
     THEN
     FtToCGAnglePls := FtToCGAnglePls - 2.0 * Pi;

     FtToCGAnglePlsTst := False;

     IF Cos (FtToCGAnglePls) <> 0.0
```

```
THEN
BEGIN

  TstLineSlope := Sin (FtToCGAnglePls) / Cos (FtToCGAnglePls);

  IF FtToCGAnglePls < Pi / 2.0
  THEN
  BEGIN

    FOR OtherLeg := FtLt TO RrRt DO
    IF OtherLeg <> Leg
    THEN
    IF VehclLegsStts [OtherLeg]. SptStt = Support
    THEN
    FtToCGAnglePlsTst := FtToCGAnglePlsTst OR
    (VehclLegsStts [OtherLeg]. PosInEarthCrd [EarthY] >=
    VehclLegsStts [Leg]. PosInEarthCrd [EarthY] +
    TstLineSlope *
    (VehclLegsStts [OtherLeg]. PosInEarthCrd [EarthX] -
    VehclLegsStts [Leg]. PosInEarthCrd [EarthX]))

  END
  ELSE
  BEGIN

    FOR OtherLeg := FtLt TO RrRt DO
    IF OtherLeg <> Leg
    THEN
    IF VehclLegsStts [OtherLeg]. SptStt = Support
    THEN
    FtToCGAnglePlsTst := FtToCGAnglePlsTst OR
    (VehclLegsStts [OtherLeg]. PosInEarthCrd [EarthY] <=
    VehclLegsStts [Leg]. PosInEarthCrd [EarthY] +
    TstLineSlope *
    (VehclLegsStts [OtherLeg]. PosInEarthCrd [EarthX] -
    VehclLegsStts [Leg]. PosInEarthCrd [EarthX]))

  END
END
ELSE
BEGIN

  IF Sin (FtToCGAnglePls) > 0.0
  THEN
  BEGIN

    FOR OtherLeg := FtLt TO RrRt DO
    IF OtherLeg <> Leg
    THEN
    IF VehclLegsStts [OtherLeg]. SptStt = Support
    THEN
    FtToCGAnglePlsTst := FtToCGAnglePlsTst OR
    (VehclLegsStts [OtherLeg]. PosInEarthCrd [EarthX] <= 0.0)

  END
  ELSE
  BEGIN

    FOR OtherLeg := FtLt TO RrRt DO
    IF OtherLeg <> Leg
    THEN
    IF VehclLegsStts [OtherLeg]. SptStt = Support
    THEN
    FtToCGAnglePlsTst := FtToCGAnglePlsTst OR
    (VehclLegsStts [OtherLeg]. PosInEarthCrd [EarthX] >= 0.0)

  END
END;

FtToCGAngleMns := FtToCGAngle + FtToCGStabMrgnArc;
IF FtToCGAngleMns < - Pi / 2.0
THEN
FtToCGAngleMns := FtToCGAngleMns + 2.0 * Pi;

FtToCGAngleMnsTst := False;

IF Cos (FtToCGAngleMns) <> 0.0
THEN
BEGIN

  TstLineSlope := Sin (FtToCGAngleMns) / Cos (FtToCGAngleMns);

  IF FtToCGAngleMns < Pi / 2.0
  THEN
  BEGIN

    FOR OtherLeg := FtLt TO RrRt DO
    IF OtherLeg <> Leg
    THEN
    IF VehclLegsStts [OtherLeg]. SptStt = Support
    THEN
    FtToCGAngleMnsTst := FtToCGAngleMnsTst OR
    (VehclLegsStts [OtherLeg]. PosInEarthCrd [EarthY] <=
    VehclLegsStts [Leg]. PosInEarthCrd [EarthY] +
    TstLineSlope *
    (VehclLegsStts [OtherLeg]. PosInEarthCrd [EarthX] -
    VehclLegsStts [Leg]. PosInEarthCrd [EarthX]))

  END
  ELSE
  BEGIN

    FOR OtherLeg := FtLt TO RrRt DO
    IF OtherLeg <> Leg
    THEN
    IF VehclLegsStts [OtherLeg]. SptStt = Support
    THEN
    FtToCGAngleMnsTst := FtToCGAngleMnsTst OR
    (VehclLegsStts [OtherLeg]. PosInEarthCrd [EarthY] >=
    VehclLegsStts [Leg]. PosInEarthCrd [EarthY] +
    TstLineSlope *
    (VehclLegsStts [OtherLeg]. PosInEarthCrd [EarthX] -
    VehclLegsStts [Leg]. PosInEarthCrd [EarthX]))

  END
END
ELSE
BEGIN

  IF Sin (FtToCGAngleMns) > 0.0
  THEN
  BEGIN
```

```
            FOR OtherLeg := FtLt TO RrRt DO
            IF OtherLeg <> Leg
            THEN
            IF VenclLegsStts [OtherLeg]. SptStt = Support
            THEN
            FtToCGAngleMnsTst := FtToCGAngleMnsTst OR
            (VenclLegsStts [OtherLeg]. PosInEarthCrd [EarthX] >= 0.0)

          END
          ELSE
          BEGIN

            FOR OtherLeg := FtLt TO RrRt DO
            IF OtherLeg <> Leg
            THEN
            IF VenclLegsStts [OtherLeg]. SptStt = Support
            THEN
            FtToCGAngleMnsTst := FtToCGAngleMnsTst OR
            (VenclLegsStts [OtherLeg]. PosInEarthCrd [EarthX] <= 0.0)

          END
        END;

        VenclIsStable := FtToCGAnglePlsTst AND FtToCGAngleMnsTst

      END
    END
  END;


  FUNCTION LegFootholdFound (
  CurrntBdySttIdx: Integer;
  VenclBdyTraj: VenclBdyTrajType;
  Leg: Legs;
  LegPosInEarthCrd: PtInEarthCrd;
  VAR VenclLegTraj: VenclLegTrajType):
  Boolean;

    CONST

      MaxVenclXYIdx =   4;
      MinVenclXIdx  = - 4;
      MaxVenclXIdx  =   1;
      MinVenclYIdx  = - 2;
      MaxVenclYIdx  =   1;

    VAR

      CurrentTime: Real;
      MnlNxtFhdInVenclCrd, PttlNxtFhdInVenclCrd: PtInVenclCrd;
      CurrentFhdInEarthCrd, PttlNxtFhdInEarthCrd: PtInEarthCrd;
      LegRtnEarthXDst, LegRtnEarthYDst: Real;
      LegRtnEarthXYDst: Real;
      LegRtnEarthXYIntrvl, LegRtnEarthXYIntrvls: Integer;
      LegRtnEarthXYDstInc: Real;
      LegRtnEarthXDstInc, LegRtnEarthYDstInc: Real;
      LftElev, PlcElev: Real;
      EarthXPos, EarthYPos, EarthZPos: Real;
      TstCurrentFhdInEarthCrd, TstPttlNxtFhdInEarthCrd: PtInEarthCrd;
      VenclXYIdx, VenclXIdx, VenclYIdx: Integer;
    FUNCTION FutureBdySttIdx (
    VenclBdyTraj: VenclBdyTrajType;
    FutureTime: Real):
    Integer;

    BEGIN

      FutureBdySttIdx := 0;

      WHILE
      (FutureBdySttIdx < VenclBdyTraj. MaxBdySttIdx) AND
      (FutureTime > VenclBdyTraj. VenclBdyStts [FutureBdySttIdx]. Time) DO
      FutureBdySttIdx := FutureBdySttIdx + 1

    END;


  BEGIN

    CurrentTime := VenclBdyTraj. VenclBdyStts [CurrntBdySttIdx]. Time;

    CASE Leg OF

      FtLt, FtRt:
      NmnlNxtFhdInVenclCrd [VenclX] := VenclCtrToVenclFtLegs + 2.0;

      CrLt, CrRt:
      NmnlNxtFhdInVenclCrd [VenclX] := VenclCtrToVenclCrLegs + 2.0;

      RrLt, RrRt:
      NmnlNxtFhdInVenclCrd [VenclX] := VenclCtrToVenclRrLegs + 2.0

    END;

    CASE Leg OF

      FtLt, CrLt, RrLt:
      IF Leg = CrLt
      THEN
      NmnlNxtFhdInVenclCrd [VenclY] := VenclCtrToVenclLtLegs + 1.25
      ELSE
      NmnlNxtFhdInVenclCrd [VenclY] := VenclCtrToVenclLtLegs + 0.625;

      FtRt, CrRt, RrRt:
      IF Leg = CrRt
      THEN
      NmnlNxtFhdInVenclCrd [VenclY] := VenclCtrToVenclRtLegs - 1.25
      ELSE
      NmnlNxtFhdInVenclCrd [VenclY] := VenclCtrToVenclRtLegs - 0.625

    END;

    NmnlNxtFhdInVenclCrd [VenclZ] := 0.0;

    LegFootholdFound := False;

    FOR VenclXYIdx := 0 TO MaxVenclXYIdx DO
    FOR VenclYIdx := - VenclXYIdx TO VenclXYIdx DO
    IF (Abs (VenclYIdx) = VenclXYIdx) AND
```

```
(VenclYIdx >= MinVenclYIdx) AND (VenclYIdx <= MaxVenclYIdx)
THEN
FOR VenclXIdx := - VenclXYIdx TO VenclXYIdx DO
IF (Abs (VenclXIdx) = VenclXYIdx) AND
(VenclXIdx >= MinVenclXIdx) AND (VenclXIdx <= MaxVenclXIdx)
THEN
IF NOT LegFootholdFound
THEN
BEGIN

  PttlNxtFhdInVehclCrd [VenclX] :=
  NmnlNxtFhdInVenclCrd [VenclX] + VenclXIdx * 1.0;

  IF Odd (Ord (Leg))
  THEN
  PttlNxtFhdInVehclCrd [VenclY] :=
  NmnlNxtFhdInVehclCrd [VenclY] - VenclYIdx * 1.0
  ELSE
  PttlNxtFhdInVehclCrd [VenclY] :=
  NmnlNxtFhdInVehclCrd [VenclY] + VenclYIdx * 1.0;

  PttlNxtFhdInVenclCrd [VenclZ] :=
  NmnlNxtFhdInVenclCrd [VenclZ];

  TrnsfrmPtToEarthCrdFrVenclCrd (PttlNxtFhdInEarthCrd,
  VenclBdyTraj. VenclBdyStts [FutureBdySttIdx (
  VenclBdyTraj, CurrentTime + LegRtnTim)].
  VenclToEarthTransMatrix, PttlNxtFhdInVehclCrd);

  PttlNxtFhdInEarthCrd [EarthZ] := TrrnHtnd (
  PttlNxtFhdInEarthCrd [EarthX], PttlNxtFhdInEarthCrd [EarthY]);

  IF PttlNxtFhdInEarthCrd [EarthZ] <> NoDatReal
  THEN
  IF FtInLimits (Leg, PttlNxtFhdInEarthCrd,
  VenclBdyTraj. VenclBdyStts [FutureBdySttIdx (
  VenclBdyTraj, CurrentTime + LegRtnTim)].
  VenclToEarthTransMatrix)
  THEN
  BEGIN

    CurrentFhdInEarthCrd [EarthX] :=
    VehclLegsStts [Leg]. PosInEarthCrd [EarthX];
    CurrentFhdInEarthCrd [EarthY] :=
    VehclLegsStts [Leg]. PosInEarthCrd [EarthY];
    CurrentFhdInEarthCrd [EarthZ] :=
    VehclLegsStts [Leg]. PosInEarthCrd [EarthZ];

    LegRtnEarthXDst :=
    PttlNxtFhdInEarthCrd [EarthX] - CurrentFhdInEarthCrd [EarthX];
    LegRtnEarthYDst :=
    PttlNxtFhdInEarthCrd [EarthY] - CurrentFhdInEarthCrd [EarthY];

    LegRtnEarthXYDst :=
    SqRt (Sqr (LegRtnEarthXDst) + Sqr (LegRtnEarthYDst));

    IF LegRtnEarthXYDst <> 0.0
    THEN
    BEGIN

      LegRtnEarthXYIntrvls := 0;
```

```
REPEAT

  LegRtnEarthXYIntrvls := LegRtnEarthXYIntrvls + 1;
  LegRtnEarthXYDstInc := LegRtnEarthXYDst / LegRtnEarthXYIntrvls

UNTIL LegRtnEarthXYDstinc <= 0.5;

LegRtnEarthXDstInc :=
LegRtnEarthXYDstInc * LegRtnEarthXDst / LegRtnEarthXYDst;
LegRtnEarthYDstInc :=
LegRtnEarthXYDstInc * LegRtnEarthYDst / LegRtnEarthXYDst;

LftElev := CurrentFhdInEarthCrd [EarthZ];
PlcElev := PttlNxtFndInEarthCrd [EarthZ];

FOR LegRtnEarthXYIntrvl := 1 TO (LegRtnEarthXYIntrvls - 1) DO
BEGIN

  EarthXPos := CurrentFndInEarthCrd [EarthX] +
  LegRtnEarthXYIntrvl * LegRtnEarthXDstInc;
  EarthYPos := CurrentFndInEarthCrd [EarthY] +
  LegRtnEarthXYIntrvl * LegRtnEarthYDstInc;

  EarthZPos := TrrnElev (EarthXPos, EarthYPos);

  IF EarthZPos <> NoDatReal
  THEN
  IF (EarthZPos > LftElev) AND (EarthZPos > PlcElev)
  THEN
  BEGIN

    LftElev := EarthZPos;
    PlcElev := EarthZPos

  END
  ELSE
  IF
  EarthZPos > (LftElev + (PlcElev - LftElev) *
  LegRtnEarthXYIntrvl / LegRtnEarthXYIntrvls)
  THEN
  IF EarthZPos > LftElev
  THEN
  LftElev := PlcElev + (EarthZPos - PlcElev) *
  LegRtnEarthXYIntrvls /
  (LegRtnEarthXYIntrvls - LegRtnEarthXYIntrvl)
  ELSE
  PlcElev := LftElev + (EarthZPos - LftElev) *
  LegRtnEarthXYIntrvls / LegRtnEarthXYIntrvl

END;

TstCurrentFndInEarthCrd := CurrentFhdInEarthCrd;
TstCurrentFndInEarthCrd [EarthZ] := LftElev + 0.5;

TstPttlNxtFndInEarthCrd := PttlNxtFndInEarthCrd;
TstPttlNxtFndInEarthCrd [EarthZ] := PlcElev + 0.5;

LegFootholdFound :=
FtInLimits (Leg, TstCurrentFndInEarthCrd,
VehclBdyTraj. VehclBdyStts [FutureBdySttIdx (
        VehclBdyTraj, CurrentTime + LegLftTim)].
        VehclToEarthTransMatrix) AND
        FtInLimits (Leg, TstPtToNxtFndInEarthCrd,
        VehclBdyTraj. VehclBdyStts [FutureBdySttIdx (
        VehclBdyTraj, CurrentTime + LegRtnTim - LegPlcTim)].
        VehclToEarthTransMatrix);

        IF LegFootholdFound
        THEN
        WITH VehclLegTraj DO
        BEGIN

          LftTime := CurrentTime;
          LftHgt  := TstCurrentFndInEarthCrd [EarthZ] -
          CurrentFndInEarthCrd [EarthZ];
          PlcTime := CurrentTime + LegRtnTim - LegPlcTim;
          TrnsfrmPtToVenclCrdFrEarthCrd (PlcPosInVenclCrd,
          VenclBdyTraj. VehclBdyStts [FutureBdySttIdx (
          VehclBdyTraj, CurrentTime + LegRtnTim - LegPlcTim)].
          VehclToEarthTransMatrix, PtToNxtFndInEarthCrd);
          CttTime := CurrentTime + LegRtnTim;
          CttHgtMin := TstPtToNxtFndInEarthCrd [EarthZ] -
          PtToNxtFndInEarthCrd [EarthZ];
          CttHgtMax := TstPtToNxtFndInEarthCrd [EarthZ] -
          PtToNxtFndInEarthCrd [EarthZ];
          NxtFndInEarthCrd := PtToNxtFndInEarthCrd;

        END
      END
    END
  END
END;


FUNCTION LegOutOfLimitsIdx (
CurrntBdySttIdx: Integer;
VenclBdyTraj: VehclBdyTrajType;
Leg: legs;
LegPosInEarthCrd: PtInEarthCrd):
Integer;

  VAR

    FtOutOfLimits: Boolean;


BEGIN

  LegOutOfLimitsIdx := CurrntBdySttIdx - 1;

  REPEAT

    LegOutOfLimitsIdx := LegOutOfLimitsIdx + 1;

    FtOutOfLimits := NOT FtInLimits (
    Leg, LegPosInEarthCrd,
    VehclBdyTraj. VehclBdyStts [LegOutOfLimitsIdx].
    VenclToEarthTransmatrix)

  UNTIL FtOutOfLimits OR (LegOutOfLimitsIdx = VenclBdyTraj. MaxBdySttIdx);
```

```
      IF NOT FtOutOfLimits
      THEN
      LegOutOfLimitsIdx := MaxInt

    END;


  BEGIN

    FutureVehclLegsStts := VehclLegsStts;

    CalcVehclLegsTrajs := True;

    FOR Leg := FtLt TO RrRt DO
    BEGIN

      LegTrajIdx [Leg] :=  0;

      IF
      (FutureVehclLegsStts [Leg]. SptStt = Support) AND
      (VehclLegsCmnds [Leg]. SptStt = Support)
      THEN
      BEGIN

        VehclLegsTrajs [Leg]. MaxLegTrajIdx := -1;

        WITH VehclLegsTrajs [Leg]. VehclLegTrajs [0] DO
        BEGIN

          LftTime := 0.0;
          LftHgt  := 0.0;
          PlcTime := 0.0;
          FOR VehclCrdIdx := VehclX TO VehclZ DO
          PlcPosInVehclCrd [VehclCrdIdx] := 0.0;
          CttTime := 0.0;
          CttHgtMin := 0.0;
          CttHgtMax := 0.0;
          FOR EarthCrdIdx := EarthX TO EarthZ DO
          NxtFndInEarthCrd [EarthCrdIdx] := 0.0

        END
      END
      ELSE
      VehclLegsTrajs [Leg]. MaxLegTrajIdx := 0

    END;

    FOR Leg := RrRt DOWNTO FtLt DO
    IF FutureVehclLegsStts [Leg]. SptStt = Support
    THEN
    LegsOutOfLimitsIdxs [Leg] :=
    LegOutOfLimitsIdx (0, VehclBdyTraj,
    Leg, FutureVehclLegsStts [Leg]. PosInEarthCrd)
    ELSE
    LegsOutOfLimitsIdxs [Leg] := MaxInt;

    BdySttIdx := - 1;

    WHILE CalcVehclLegsTrajs AND (BdySttIdx < VehclBdyTraj. MaxBdySttIdx) DO
    BEGIN
```

```
BdySttIdx := BdySttIdx + 1;

FutureTime := VenclBdyTraj. VehclBdyStts [BdySttIdx]. Time;

FOR Leg := RrRt DOWNTO FtLt DO
BEGIN

  IF (LegTrajIdx [Leg] <= VenclLegsTrajs [Leg]. MaxLegTrajIdx)
  THEN
  IF (FutureVenclLegsStts [Leg]. SptStt = Support)
  THEN
  IF
  (FutureTime >=
  VenclLegsTrajs [Leg]. VehclLegTrajs [LegTrajIdx [Leg]]. LftTime)
  THEN
  FutureVehclLegsStts [Leg]. SptStt := Trnsfer;

  IF (FutureVehclLegsStts [Leg]. SptStt = Trnsfer)
  THEN
  IF
  (FutureTime >=
  VehclLegsTrajs [Leg]. VehclLegTrajs [LegTrajIdx [Leg]]. CttTime)
  THEN
  BEGIN

    FutureVenclLegsStts [Leg]. SptStt := Support;
    FutureVehclLegsStts [Leg]. PosInEarthCrd :=
    VehclLegsTrajs [Leg]. VenclLegTrajs [LegTrajIdx [Leg]].
    NxtHndInEarthCrd;

    LegTrajIdx [Leg] := LegTrajIdx [Leg] + 1;

    LegsOutOfLimitsIdxs [Leg] := LegOutOfLimitsIdx (
    BdySttIdx, VenclBdyTraj,
    Leg, FutureVehclLegsStts [Leg]. PosInEarthCrd)

  END
END;

CalcVenclLegsTrajs := VehclIsStable (
VenclBdyTraj. VehclBdyStts [BdySttIdx]. VenclToEarthTransMatrix,
FutureVehclLegsStts);

IF CalcVenclLegsTrajs
THEN
BEGIN

  PotentialVenclLegsStts := FutureVenclLegsStts;

  LegsCanBeLifted := True;

  REPEAT

    MinLegOutOfLimitsIdx := MaxInt;

    FOR Leg := RrRt DOWNTO FtLt DO
    IF LegsOutOfLimitsIdxs [Leg] < MinLegOutOfLimitsIdx
    THEN
    BEGIN

      MinLegOutOfLimitsIdx := LegsOutOfLimitsIdxs [Leg];
```

```
            LegMinLegOutOfLimits := Leg

          END;

          IF MinLegOutOfLimitsIdx < MaxInt
          THEN
          BEGIN

            PotentialVehclLegsStts [Leg]. SptStt := Trnsfer;

            IF VehclIsStable (
            VehclBodyTraj. VehclBdyStts [BdySttIdx]. VehclToEarthTransMatrix,
            PotentialVehclLegsStts)
            THEN
            IF LegFootholdFound (
            BdySttIdx, VehclBdyTraj,
            Leg, PotentialVehclLegsStts [Leg]. PosInEarthCrd,
            VehclLegTraj)
            THEN
            BEGIN

              VehclLegsTrajs [Leg]. MaxLegTrajIdx :=
              VehclLegsTrajs [Leg]. MaxLegTrajIdx + 1;

              VehclLegsTrajs [Leg].
              VehclLegTrajs [VehclLegsTrajs [Leg]. MaxLegTrajIdx] :=
              VehclLegTraj;

              LegsOutOfLimitsIdxs [Leg] := MaxInt

            END
            ELSE
            BEGIN

              PotentialVehclLegsStts [Leg]. SptStt := Support;

              LegsCanBeLifted := False

            END
            ELSE
            BEGIN

              PotentialVehclLegsStts [Leg]. SptStt := Support;

              LegsCanBeLifted := False

            END;

            CalcVehclLegsTrajs := NOT
            ((MinLegOutOfLimitsIdx = BdySttIdx) AND (NOT LegsCanBeLifted))

          END

        UNTIL (MinLegOutOfLimitsIdx = MaxInt) OR (NOT LegsCanBeLifted)

      END
    END
  END;

BEGIN
```

```
Brd4InOperationPtr. SegAddr := 8190;
Brd4InOperationPtr. OffAddr :=    0;

Brd4InOperationPtr. AbsAddr^ := False;

Brd1InOperationPtr. SegAddr := 8190;
Brd1InOperationPtr. OffAddr :=    1;

Brds14CommIdleBufferBusyFromBrd4Ptr. SegAddr := 8189;
Brds14CommIdleBufferBusyFromBrd4Ptr. OffAddr :=    1;
Brds14CommIdleBufferBusyFromBrd1Ptr. SegAddr := 8189;
Brds14CommIdleBufferBusyFromBrd1Ptr. OffAddr :=    0;

Brds14CommIdleBufferBusyFromBrd4Ptr. AbsAddr^ := False;

Brds14CommOtptBufferPtrFromBrd4Ptr. SegAddr := 8188;
Brds14CommOtptBufferPtrFromBrd4Ptr. OffAddr :=    8;
Brds14CommIdleBufferPtrFromBrd4Ptr. SegAddr := 8188;
Brds14CommIdleBufferPtrFromBrd4Ptr. OffAddr :=    4;
Brds14CommInptBufferPtrFromBrd4Ptr. SegAddr := 8188;
Brds14CommInptBufferPtrFromBrd4Ptr. OffAddr :=    0;
Brds14CommOtptBufferPtrFromBrd1Ptr. SegAddr := 8187;
Brds14CommOtptBufferPtrFromBrd1Ptr. OffAddr :=    8;
Brds14CommIdleBufferPtrFromBrd1Ptr. SegAddr := 8187;
Brds14CommIdleBufferPtrFromBrd1Ptr. OffAddr :=    4;
Brds14CommInptBufferPtrFromBrd1Ptr. SegAddr := 8187;
Brds14CommInptBufferPtrFromBrd1Ptr. OffAddr :=    0;

Brds41CommIdleBufferBusyFromBrd4Ptr. SegAddr := 8096;
Brds41CommIdleBufferBusyFromBrd4Ptr. OffAddr :=    1;
Brds41CommIdleBufferBusyFromBrd1Ptr. SegAddr := 8096;
Brds41CommIdleBufferBusyFromBrd1Ptr. OffAddr :=    0;

Brds41CommOtptBufferPtrFromBrd4Ptr. SegAddr := 8095;
Brds41CommOtptBufferPtrFromBrd4Ptr. OffAddr :=    8;
Brds41CommIdleBufferPtrFromBrd4Ptr. SegAddr := 8095;
Brds41CommIdleBufferPtrFromBrd4Ptr. OffAddr :=    4;
Brds41CommInptBufferPtrFromBrd4Ptr. SegAddr := 8095;
Brds41CommInptBufferPtrFromBrd4Ptr. OffAddr :=    0;
Brds41CommOtptBufferPtrFromBrd1Ptr. SegAddr := 8094;
Brds41CommOtptBufferPtrFromBrd1Ptr. OffAddr :=    8;
Brds41CommIdleBufferPtrFromBrd1Ptr. SegAddr := 8094;
Brds41CommIdleBufferPtrFromBrd1Ptr. OffAddr :=    4;
Brds41CommInptBufferPtrFromBrd1Ptr. SegAddr := 8094;
Brds41CommInptBufferPtrFromBrd1Ptr. OffAddr :=    0;

Brds41CommOtptBufferPtrFromBrd4Ptr. AbsAddr^. SegAddr :=  7711;
Brds41CommOtptBufferPtrFromBrd4Ptr. AbsAddr^. OffAddr :=     0;
Brds41CommIdleBufferPtrFromBrd4Ptr. AbsAddr^. SegAddr :=  7211;
Brds41CommIdleBufferPtrFromBrd4Ptr. AbsAddr^. OffAddr :=     0;
Brds41CommInptBufferPtrFromBrd4Ptr. AbsAddr^. SegAddr :=  6711;
Brds41CommInptBufferPtrFromBrd4Ptr. AbsAddr^. OffAddr :=     0;
Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. SegAddr := -8673;
Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. OffAddr :=     0;
Brds41CommIdleBufferPtrFromBrd1Ptr. AbsAddr^. SegAddr := -9173;
Brds41CommIdleBufferPtrFromBrd1Ptr. AbsAddr^. OffAddr :=     0;
Brds41CommInptBufferPtrFromBrd1Ptr. AbsAddr^. SegAddr := -9673;
Brds41CommInptBufferPtrFromBrd1Ptr. AbsAddr^. OffAddr :=     0;

Brds41CommIdleBufferPtrFromBrd4Ptr. AbsAddr^. AbsAddr^. NewData := False;
```

```
Brds14CommOtptBufferPtrFromBrd4Ptr. AbsAddr^ :=
Brds14CommIdleBufferPtrFromBrd4Ptr. AbsAddr^;

Brds14CommIdleBufferPtrFromBrd4Ptr. AbsAddr^ :=
Brds14CommTempBufferPtr;

Brds14CommIdleBufferPtrFromBrd4Ptr. AbsAddr^. AbsAddr^. NewData :=
False;

Brds14CommIdleBufferBusyFromBrd4Ptr. AbsAddr^ := False;

WITH Brds14CommOtptBufferPtrFromBrd4Ptr. AbsAddr^. AbsAddr^ DO
BEGIN

  TransVelRqstMag := SqRt (Sqr (FrwdVelRqst) + Sqr (SideVelRqst));

  IF TransVelRqstMag <= VehclMaxTransVel
  THEN
  VelRedFctr := 1.0
  ELSE
  VelRedFctr := VehclMaxTransVel / TransVelRqstMag;

  IF TurnVelRqst > VehclMaxRotatVel
  THEN
  IF (VehclMaxRotatVel / TurnVelRqst) < VelRedFctr
  THEN
  VelRedFctr := VehclMaxRotatVel / TurnVelRqst;

  TransAccRqstMag := SqRt (
  Sqr (FrwdVelRqst * VelRedFctr - VehclLinVelInVehclCrd [VehclX]) +
  Sqr (SideVelRqst * VelRedFctr - VehclLinVelInVehclCrd [VehclY])) /
  GuidAlgrthmExecIntrvl;

  IF TransAccRqstMag <= VehclMaxTransAcc
  THEN
  AccRedFctr := 1.0
  ELSE
  AccRedFctr := VehclMaxTransAcc / TransAccRqstMag;

  RotatAccRqstMag :=
  (TurnVelRqst * VelRedFctr - VehclAngVelInVehclCrd [VehclZ]) /
  GuidAlgrthmExecIntrvl;

  IF RotatAccRqstMag <= VehclMaxRotatAcc
  THEN
  IF (VehclMaxRotatAcc / RotatAccRqstMag) < AccRedFctr
  THEN
  AccRedFctr := VehclMaxRotatAcc / RotatAccRqstMag;

  Brds41CommInptBufferPtrFromBrd4Ptr. AbsAddr^. AbsAddr^.
  NewData := False;

  IF CalcVehclBdyTraj (
  1.0 * AccRedFctr * VelRedFctr * FrwdVelRqst / GuidAlgrthmExecIntrvl,
  1.0 * AccRedFctr * VelRedFctr * SideVelRqst / GuidAlgrthmExecIntrvl,
  1.0 * AccRedFctr * VelRedFctr * TurnVelRqst / GuidAlgrthmExecIntrvl,
  CurrntTime, VehclToEarthTransMatrix,
  VehclLinVelInVehclCrd, VehclAngVelInVehclCrd,
  VehclBdyTraj)
  THEN
  IF CalcVehclLegsTrajs (
VehclBdyTraj, VehclLegsStts, VehclLegsCmnds,
VehclLegsTrajs)
THEN
BEGIN

  Brds41CommInptBufferPtrFromBrd4Ptr. AbsAddr^. AbsAddr^.
  VehclBdyTraj := VehclBdyTraj;
  Brds41CommInptBufferPtrFromBrd4Ptr. AbsAddr^. AbsAddr^.
  VehclLegsTrajs := VehclLegsTrajs;
  Brds41CommInptBufferPtrFromBrd4Ptr. AbsAddr^. AbsAddr^.
  NewData := True

END
ELSE
IF CalcVehclBdyTraj (
0.3 * AccRedFctr * VelRedFctr * FrwdVelRqst / GuidAlgrthmExecIntrvl,
0.3 * AccRedFctr * VelRedFctr * SideVelRqst / GuidAlgrthmExecIntrvl,
0.3 * AccRedFctr * VelRedFctr * TurnVelRqst / GuidAlgrthmExecIntrvl,
CurrntTime, VehclToEarthTransMatrix,
VehclLinVelInVehclCrd, VehclAngVelInVehclCrd,
VehclBdyTraj)
THEN
IF CalcVehclLegsTrajs (
VehclBdyTraj, VehclLegsStts, VehclLegsCmnds,
VehclLegsTrajs)
THEN
BEGIN

  Brds41CommInptBufferPtrFromBrd4Ptr. AbsAddr^. AbsAddr^.
  VehclBdyTraj := VehclBdyTraj;
  Brds41CommInptBufferPtrFromBrd4Ptr. AbsAddr^. AbsAddr^.
  VehclLegsTrajs := VehclLegsTrajs;
  Brds41CommInptBufferPtrFromBrd4Ptr. AbsAddr^. AbsAddr^.
  NewData := True

END
ELSE
IF CalcVehclBdyTraj (
0.1 * AccRedFctr * VelRedFctr * FrwdVelRqst / GuidAlgrthmExecIntrvl,
0.1 * AccRedFctr * VelRedFctr * SideVelRqst / GuidAlgrthmExecIntrvl,
0.1 * AccRedFctr * VelRedFctr * TurnVelRqst / GuidAlgrthmExecIntrvl,
CurrntTime, VehclToEarthTransMatrix,
VehclLinVelInVehclCrd, VehclAngVelInVehclCrd,
VehclBdyTraj)
THEN
IF CalcVehclLegsTrajs (
VehclBdyTraj, VehclLegsStts, VehclLegsCmnds,
VehclLegsTrajs)
THEN
BEGIN

  Brds41CommInptBufferPtrFromBrd4Ptr. AbsAddr^. AbsAddr^.
  VehclBdyTraj := VehclBdyTraj;
  Brds41CommInptBufferPtrFromBrd4Ptr. AbsAddr^. AbsAddr^.
  VehclLegsTrajs := VehclLegsTrajs;
  Brds41CommInptBufferPtrFromBrd4Ptr. AbsAddr^. AbsAddr^.
  NewData := True

END;

IF Brds41CommInptBufferPtrFromBrd4Ptr. AbsAddr^. AbsAddr^. NewData
```

```
      THEN
      BEGIN

        REPEAT

          Brds41CommIdleBufferBusyFromBrd4Ptr. AbsAddr^ := True;

          IF Brds41CommIdleBufferBusyFromBrd1Ptr. AbsAddr^
          THEN
          Brds41CommIdleBufferBusyFromBrd4Ptr. AbsAddr^ := False

        UNTIL Brds41CommIdleBufferBusyFromBrd4Ptr. AbsAddr^;

        Brds41CommTempBufferPtr :=
        Brds41CommInptBufferPtrFromBrd4Ptr. AbsAddr^;

        Brds41CommInptBufferPtrFromBrd4Ptr. AbsAddr^ :=
        Brds41CommIdleBufferPtrFromBrd4Ptr. AbsAddr^;

        Brds41CommIdleBufferPtrFromBrd4Ptr. AbsAddr^ :=
        Brds41CommTempBufferPtr;

        Brds41CommTempBufferPtr :=
        Brds41CommInptBufferPtrFromBrd1Ptr. AbsAddr^;

        Brds41CommInptBufferPtrFromBrd1Ptr. AbsAddr^ :=
        Brds41CommIdleBufferPtrFromBrd1Ptr. AbsAddr^;

        Brds41CommIdleBufferPtrFromBrd1Ptr. AbsAddr^ :=
        Brds41CommTempBufferPtr;

        Brds41CommIdleBufferPtrFromBrd4Ptr. AbsAddr^. AbsAddr^. NewData := True;

        Brds41CommIdleBufferBusyFromBrd4Ptr. AbsAddr^ := False;

        Brd1InOperationPtr. AbsAddr^ := True

      END
    END
  END
END
```

APPENDIX G

VEHICLE CONTROL COMMUNICATION BOARD PROGRAM LISTING

```
MODULE VehicleStateAndCommandCommunication;

PROGRAM VehicleStateAndCommandCommunication (Input, Output);

  CONST

    Pi = 3.141592654;

    RealIInteger = 63;
    RealIntegerPlus1 = 64;

    ScnnrXOfst = 6.000667;
    ScnnrYOfst = 0.0;
    ScnnrZOfst = 1.0;

    MaxBodySttsInBdyTraj = 60;
    MaxLegTrajsInLegTrajsRec = 10;

    NoTraj = -1;

  TYPE

    Bits = (Bit0, Bit1, Bit2, Bit3. Bit4, Bit5, Bit6, Bit7);
    SetOfBits = SET OF Bits;

    Byte =
    RECORD
      Case Integer OF
        0: (Chrctr: Char);
        1: (BitSet: SetOfBits)
    END;

    TwoCharInteger =
    RECORD
      CASE Integer OF
        0: (LoChar,
             HiChar: Char);
        1: (IntVal: Integer)
    END;

    Crd = (X, Y, Z);
    PtInCrd = ARRAY [Crd] OF Real;
    VcInCrd = ARRAY [Crd] OF Real;

    EarthCrd = (EarthX, EarthY, EarthZ);
    PtInEarthCrd = ARRAY [EarthCrd] OF Real;
    VcInEarthCrd = ARRAY [EarthCrd] OF Real;

    VehclCrd = (VehclX, VehclY, VehclZ);
    PtInVehclCrd = ARRAY [VehclCrd] OF Real;
    VcInVehclCrd = ARRAY [VehclCrd] OF Real;

    EulerAngles = (Yaw, Pch, Rll);
    OrInEulerAngles = ARRAY [EulerAngles] OF Real;

    DegOfFrdm =
    RECORD
      Rotat: OrInEulerAngles;
      Trans: PtInCrd
```

```
END;

TransMatrix =
RECORD
  Rotat: ARRAY [Crd] OF VcInCrd;
  Trans: PtInCrd
END;

TransMatrixInteger =
RECORD
  Rotat: ARRAY [Crd] OF ARRAY [Crd] OF Integer;
  Trans: ARRAY [Crd] OF Integer
END;

Legs = (None, FtLt, FtRt, CrLt, CrRt, RrLt, RrRt);

VenclBdyStt =
RECORD
  Time: Real;
  VenclToEarthTransMatrix: TransMatrix
END;

VenclBdyTrajType =
RECORD
  MaxBdySttIdx: Integer;
  VenclBdyStts: ARRAY [0..MaxBdySttsInBdyTraj] OF VenclBdyStt
END;

SptSttType = (Trnsfer, Support);

VenclLegStt =
RECORD
  SptStt: SptSttType;
  PosInEarthCrd: PtInEarthCrd
END;

VenclLegsSttsType = ARRAY [Legs] OF VenclLegStt;

VenclLegTraj =
RECORD
  LftTime: Real;
  LftHgt: Real;
  PlcTime: Real;
  PlcPosInVenclCrd: PtInVenclCrd;
  CttTime: Real;
  CttHgtMin: Real;
  CttHgtMax: Real;
  NxtFhdInEarthCrd: PtInEarthCrd
END;

VenclLegTrajsRec =
RECORD
  MaxLegTrajIdx: Integer;
  VenclLegTrajs: ARRAY [0..MaxLegTrajsInLegTrajsRec] OF VenclLegTraj
END;

VenclLegsTrajsType = ARRAY [Legs] OF VenclLegTrajsRec;

VenclLegCmnd =
RECORD
  SptStt: SptSttType;
  VehclLegCmndTraj: VehclLegTraj
END;

VehclLegsCmndsType = ARRAY [Legs] OF VehclLegCmnd;

UnitVectorCompIntegerIndexedArray =
ARRAY [-RealIntegerPlus1..RealInteger] OF Integer;

BooleanPtr =
RECORD
  CASE Integer OF
    0: (AbsAddr: ^Boolean);
    1: (OffAddr,
        SegAddr: Integer)
END;

UnitVectorCompIntegerIndexedArrayPtr =
RECORD
  CASE Integer OF
    0: (AbsAddr: ^UnitVectorCompIntegerIndexedArray);
    1: (OffAddr,
        SegAddr: TwoCharInteger)
END;

Brds12CommBuffer =
RECORD
  SchnrfcEarthTransMatrix: TransMatrixInteger;
  NewData: Boolean
END;

Brds12CommBufferPtr =
RECORD
  CASE Integer OF
    0: (AbsAddr: ^Brds12CommBuffer);
    1: (OffAddr,
        SegAddr: Integer)
END;

Brds12CommBufferPtrPtr =
RECORD
  CASE Integer OF
    0: (AbsAddr: ^Brds12CommBufferPtr);
    1: (OffAddr,
        SegAddr: Integer)
END;

Brds14CommBuffer =
RECORD
  CurIntTime: Real;
  VehclToEarthTransMatrix: TransMatrix;
  VehclLinVelInVehclCrd,
  VehclAngVelInVehclCrd: VcInVehclCrd;
  VehclLegsStts: VehclLegsSttsType;
  FrwdVelRqst,
  SideVelRqst,
  TurnVelRqst: Real;
  VehclLegsCmnds: VehclLegsCmndsType;
  NewData: Boolean
END;

Brds14CommBufferPtr =
    RECORD
      CASE Integer OF
        0: (AbsAddr: ^Brds14CommBuffer);
        1: (OffAddr,
            SegAddr: Integer)
    END;

    Brds14CommBufferPtrPtr =
    RECORD
      CASE Integer OF
        0: (AbsAddr: ^Brds14CommBufferPtr);
        1: (OffAddr,
            SegAddr: Integer)
    END;

    Brds41CommBuffer =
    RECORD
      VehclBdyTraj: VehclBdyTrajType;
      VehclLegsTrajs: VehclLegsTrajsType;
      NewData: Boolean
    END;

    Brds41CommBufferPtr =
    RECORD
      CASE Integer OF
        0: (AbsAddr: ^Brds41CommBuffer);
        1: (OffAddr,
            SegAddr: Integer)
    END;

    Brds41CommBufferPtrPtr =
    RECORD
      CASE Integer OF
        0: (AbsAddr: ^Brds41CommBufferPtr);
        1: (OffAddr,
            SegAddr: Integer)
    END;

  VAR

    DummyChar: Char;
    PortCByte: Byte;
    InptSync,
    OtptSync: Boolean;
    SyncChar,
    InptSyncChar0,
    InptSyncChar1,
    InptSyncChar2,
    OtptSyncChar: Char;
    DummyReal: Real;
    ElCosProdArrayPtr,
    ElSinProdArrayPtr,
    AzCosProdArrayPtr,
    AzSinProdArrayPtr,
    ScanPtDispArrayPtr:
    UnitVectorCompIntegerIndexedArrayPtr;
    Brd2InOperationPtr,
    Brd4InOperationPtr,
    Brd1InOperationPtr: BooleanPtr;
    Brds12CommInptBufferPtrFromBrd1Ptr,
    Brds12CommIdleBufferPtrFromBrd1Ptr,
Brds12CommOtptBufferPtrFromBrd1Ptr,
Brds12CommInptBufferPtrFromBrd2Ptr,
Brds12CommIdleBufferPtrFromBrd2Ptr,
Brds12CommOtptBufferPtrFromBrd2Ptr: Brds12CommBufferPtrPtr;
Brds12CommTempBufferPtr: Brds12CommBufferPtr;
Brds12CommIdleBufferBusyFromBrd1Ptr,
Brds12CommIdleBufferBusyFromBrd2Ptr: BooleanPtr;
Brds14CommInptBufferPtrFromBrd1Ptr,
Brds14CommIdleBufferPtrFromBrd1Ptr,
Brds14CommOtptBufferPtrFromBrd1Ptr,
Brds14CommInptBufferPtrFromBrd4Ptr,
Brds14CommIdleBufferPtrFromBrd4Ptr,
Brds14CommOtptBufferPtrFromBrd4Ptr: Brds14CommBufferPtrPtr;
Brds14CommTempBufferPtr: Brds14CommBufferPtr;
Brds14CommIdleBufferBusyFromBrd1Ptr,
Brds14CommIdleBufferBusyFromBrd4Ptr: BooleanPtr;
Brds41CommInptBufferPtrFromBrd1Ptr,
Brds41CommIdleBufferPtrFromBrd1Ptr,
Brds41CommOtptBufferPtrFromBrd1Ptr,
Brds41CommInptBufferPtrFromBrd4Ptr,
Brds41CommIdleBufferPtrFromBrd4Ptr,
Brds41CommOtptBufferPtrFromBrd4Ptr: Brds41CommBufferPtrPtr;
Brds41CommTempBufferPtr: Brds41CommBufferPtr;
Brds41CommIdleBufferBusyFromBrd1Ptr,
Brds41CommIdleBufferBusyFromBrd4Ptr: BooleanPtr;
OperatingMode: Char;
Status: Char;
CurrntTime: Real;
VehclToEarthTransmatrix: Transmatrix;
BaySttIdx: Integer;
BaySttIdxFound: Boolean;
LastVehclPosDegOfFromWRTEarthCrd,
NextVehclPosDegOfFromWRTEarthCrd,
CrntVehclPosDegOfFromWRTEarthCrd: DegOfFrom;
CrntVehclVelDegOfFromWRTEarthCrd: DegOfFrom;
IntrpltnTimeRatio,
VelCalcTimeIntrvl: Real;
CrntVehclToEarthTransMatrix: TransMatrix;
VehclLinVelInEarthCrd,
VehclAngVelInEarthCrd: VcInEarthCrd;
VehclLinVelInVehclCrd,
VehclAngVelInVehclCrd: VcInVehclCrd;
VehclLinVelInEarthCrdCmnd,
VehclAngVelInEarthCrdCmnd: VcInEarthCrd;
VehclLinVelInVehclCrdCmnd,
VehclAngVelInVehclCrdCmnd: VcInVehclCrd;
LegPosInVehclCrd: PtInVehclCrd;
LegsTrajIdx: ARRAY [Legs] OF Integer;
LegTrajIdxFound: Boolean;
ScnnrPosInVehclCrd: PtInVehclCrd;
ScnnrPosInEarthCrd: PtInEarthCrd;
SptSttItgr: Integer;
LegIdx,
LegIndex: Legs;
EarthCrdIndex: EarthCrd;
CrdIndex,
CrdIndex1,
CrdIndex2: Crd;
VehclCrdIndex: VehclCrd;
ArrayIndex1,
ArrayIndex2: Integer;
```

```
  ArrayIndex2TwoChar: TwoCharInteger;
  VenclLegsSptStts: ARRAY [Legs] OF SptSttType;


PROCEDURE InReal (
VAR RealValue: Real);

  TYPE

    FourCharReal =
    RECORD
      CASE Integer OF
        0: (LowordLoChar,
            LowordHiChar,
            HiwordLoChar,
            HiwordHiChar: Char);
        1: (RealVal: Real)
    END;

  VAR

    TempRealValue: FourCharReal;


BEGIN

  REPEAT

    InByt (0CCH, PortCByte. Chrctr)

  UNTIL (PortCByte. BitSet * [Bit5]) <> [];

  InByt (0C8H, TempRealValue. LowordLoChar);

  REPEAT

    InByt (0CCH, PortCByte. Chrctr)

  UNTIL (PortCByte. BitSet * [Bit5]) <> [];

  InByt (0C8H, TempRealValue. LowordHiChar);

  REPEAT

    InByt (0CCH, PortCByte. Chrctr)

  UNTIL (PortCByte. BitSet * [Bit5]) <> [];

  InByt (0C8H, TempRealValue. HiwordLoChar);

  REPEAT

    InByt (0CCH, PortCByte. Chrctr)

  UNTIL (PortCByte. BitSet * [Bit5]) <> [];

  InByt (0C8H, TempRealValue. HiwordHiChar);

  RealValue := TempRealValue. RealVal

END;
```

```
PROCEDURE InItgr (
VAR ItgrValue: Integer);

  TYPE

    TwoCharItgr =
    RECORD
      CASE Integer OF
        0: (LoChar,
            HiChar: Char);
        1: (ItgrVal: Integer)
    END;

  VAR

    TempItgrValue: TwoCharItgr;


BEGIN

  REPEAT

    InByt (0CCH, PortCByte. Chrctr)

  UNTIL (PortCByte. BitSet * [Bit5]) <> [];

  InByt (0C8H, TempItgrValue. LoChar);

  REPEAT

    InByt (0CCH, PortCByte. Chrctr)

  UNTIL (PortCByte. BitSet * [Bit5]) <> [];

  InByt (0C8H, TempItgrValue. HiChar);

  ItgrValue := TempItgrValue. ItgrVal

END;


PROCEDURE InBool (
VAR BoolValue: Boolean);

  TYPE

    OneCharBoolean =
    RECORD
      CASE Integer OF
        0: (CharVal: Char);
        1: (BoolVal: Boolean)
    END;

  VAR

    TempBoolValue: OneCharBoolean;


BEGIN
```

```
    REPEAT

      InByt (0CCH, PortCByte. Chrctr)

    UNTIL (PortCByte. BitSet * [Bit5]) <> [];

    InByt (0C8H, TempBoolValue. CharVal);

    BoolValue := TempBoolValue. BoolVal

  END;


  PROCEDURE InChar (
  VAR CharValue: Char);


  BEGIN

    REPEAT

      InByt (0CCH, PortCByte. Chrctr)

    UNTIL (PortCByte. BitSet * [Bit5]) <> [];

    InByt (0C8H, CharValue)

  END;


  PROCEDURE OutReal (
  RealValue: Real);

    TYPE

      FourCharReal =
      RECORD
        CASE Integer OF
          0: (LowordLoChar,
              LowordHiChar,
              HiwordLoChar,
              HiwordHiChar: Char);
          1: (RealVal: Real)
      END;

    VAR

      TempRealValue: FourCharReal;


  BEGIN

    TempRealValue. RealVal := RealValue;

    REPEAT

      InByt (0CCH, PortCByte. Chrctr)

    UNTIL (PortCByte. BitSet * [Bit0]) <> [];
  OutByt (0CAH, TempRealValue. LowordLoChar);
  REPEAT
    InByt (0CCH, PortCByte. Chrctr)
  UNTIL (PortCByte. BitSet * [Bit0]) <> [];
  OutByt (0CAH, TempRealValue. LowordHiChar);
  REPEAT
    InByt (0CCH, PortCByte. Chrctr)
  UNTIL (PortCByte. BitSet * [Bit0]) <> [];
  OutByt (0CAH, TempRealValue. HiwordLoChar);
  REPEAT
    InByt (0CCH, PortCByte. Chrctr)
  UNTIL (PortCByte. BitSet * [Bit0]) <> [];
  OutByt (0CAH, TempRealValue. HiwordHiChar)
END;

PROCEDURE OutItgr (
ItgrValue: Integer);

  TYPE

    TwoCharItgr =
    RECORD
      CASE Integer OF
        0: (LoChar,
            HiChar: Char);
        1: (ItgrVal: Integer)
    END;

  VAR

    TempItgrValue: TwoCharItgr;

BEGIN

  TempItgrValue. ItgrVal := ItgrValue;

  REPEAT

    InByt (0CCH, PortCByte. Chrctr)

  UNTIL (PortCByte. BitSet * [Bit0]) <> [];

  OutByt (0CAH, TempItgrValue. LoChar);

  REPEAT
```

```
    InByt (0CCH, PortCByte. ChrCtr)

  UNTIL (PortCByte. BitSet * [bit0]) <> [];

  OutByt (0CAH, TempItgrValue. HiChar)

END;


PROCEDURE OutBool (
BoolValue: Boolean);

  TYPE

    OneCharBoolean =
    RECORD
      CASE Integer OF
        0: (CharVal: Char);
        1: (BoolVal: Boolean)
    END;

  VAR

    TempBoolValue: OneCharBoolean;


BEGIN

  TempBoolValue. BoolVal := BoolValue;

  REPEAT

    InByt (0CCH, PortCByte. ChrCtr)

  UNTIL (PortCByte. BitSet * [bit0]) <> [];

  OutByt (0CAH, TempBoolValue. CharVal)

END;


PROCEDURE OutChar (
CharValue: Char);


BEGIN

  REPEAT

    InByt (0CCH, PortCByte. ChrCtr)

  UNTIL (PortCByte. BitSet * [bit0]) <> [];

  OutByt (0CAH, CharValue)

END;


PROCEDURE TrnsfrmPtToEarthCrdFrVehclCrd (
VAR PtInEarthCrdVar: PtInEarthCrd;
VehclToEarthTransMatrix: TransMatrix;
PtInVehclCrdVar: PtInVehclCrd);

BEGIN

  PtInEarthCrdVar [EarthX] :=
  VehclToEarthTransMatrix. Rotat [X] [X] * PtInVehclCrdVar [VehclX] +
  VehclToEarthTransMatrix. Rotat [Y] [X] * PtInVehclCrdVar [VehclY] +
  VehclToEarthTransMatrix. Rotat [Z] [X] * PtInVehclCrdVar [VehclZ] +
  VehclToEarthTransMatrix. Trans [X];

  PtInEarthCrdVar [EarthY] :=
  VehclToEarthTransMatrix. Rotat [X] [Y] * PtInVehclCrdVar [VehclX] +
  VehclToEarthTransMatrix. Rotat [Y] [Y] * PtInVehclCrdVar [VehclY] +
  VehclToEarthTransMatrix. Rotat [Z] [Y] * PtInVehclCrdVar [VehclZ] +
  VehclToEarthTransMatrix. Trans [Y];

  PtInEarthCrdVar [EarthZ] :=
  VehclToEarthTransMatrix. Rotat [X] [Z] * PtInVehclCrdVar [VehclX] +
  VehclToEarthTransMatrix. Rotat [Y] [Z] * PtInVehclCrdVar [VehclY] +
  VehclToEarthTransMatrix. Rotat [Z] [Z] * PtInVehclCrdVar [VehclZ] +
  VehclToEarthTransMatrix. Trans [Z]

END;


PROCEDURE TrnsfrmVcToVehclCrdFrEarthCrd (
VAR VcInVehclCrdVar: VcInVehclCrd;
VehclToEarthTransMatrix: TransMatrix;
VcInEarthCrdVar: VcInEarthCrd);

BEGIN

  VcInVehclCrdVar [VehclX] :=
  VehclToEarthTransMatrix. Rotat [X] [X] * VcInEarthCrdVar [EarthX] +
  VehclToEarthTransMatrix. Rotat [X] [Y] * VcInEarthCrdVar [EarthY] +
  VehclToEarthTransMatrix. Rotat [X] [Z] * VcInEarthCrdVar [EarthZ];

  VcInVehclCrdVar [VehclY] :=
  VehclToEarthTransMatrix. Rotat [Y] [X] * VcInEarthCrdVar [EarthX] +
  VehclToEarthTransMatrix. Rotat [Y] [Y] * VcInEarthCrdVar [EarthY] +
  VehclToEarthTransMatrix. Rotat [Y] [Z] * VcInEarthCrdVar [EarthZ];

  VcInVehclCrdVar [VehclZ] :=
  VehclToEarthTransMatrix. Rotat [Z] [X] * VcInEarthCrdVar [EarthX] +
  VehclToEarthTransMatrix. Rotat [Z] [Y] * VcInEarthCrdVar [EarthY] +
  VehclToEarthTransMatrix. Rotat [Z] [Z] * VcInEarthCrdVar [EarthZ]

END;


PROCEDURE CalcTransMatrixFromPosDegOfFrdm (
VAR TransMatrixVar: TransMatrix;
DegOfFrdmVar: DegOfFrdm);

  VAR

    CosYaw, SinYaw,
    CosPch, SinPch,
    CosRll, SinRll: Real;

BEGIN
```

```
    CosYaw := Cos (DegOfFromVar. Rotat [Yaw]);
    SinYaw := Sin (DegOfFromVar. Rotat [Yaw]);
    CosPch := Cos (DegOfFromVar. Rotat [Pch]);
    SinPch := Sin (DegOfFromVar. Rotat [Pch]);
    CosRll := Cos (DegOfFromVar. Rotat [Rll]);
    SinRll := Sin (DegOfFromVar. Rotat [Rll]);

    TransMatrixVar. Rotat [X] [X] := CosYaw * CosPch;
    TransMatrixVar. Rotat [X] [Y] := SinYaw * CosPch;
    TransMatrixVar. Rotat [X] [Z] :=        - SinPch;

    TransMatrixVar. Rotat [Y] [X] :=
    CosYaw * SinPch * SinRll - SinYaw * CosRll;
    TransMatrixVar. Rotat [Y] [Y] :=
    SinYaw * SinPch * SinRll + CosYaw * CosRll;
    TransMatrixVar. Rotat [Y] [Z] :=
    (* *)     CosPch * SinRll;

    TransMatrixVar. Rotat [Z] [X] :=
    CosYaw * SinPch * CosRll + SinYaw * SinRll;
    TransMatrixVar. Rotat [Z] [Y] :=
    SinYaw * SinPch * CosRll - CosYaw * SinRll;
    TransMatrixVar. Rotat [Z] [Z] :=
    (* *)     CosPch * CosRll;

    TransMatrixVar. Trans [X] := DegOfFromVar. Trans [X];
    TransMatrixVar. Trans [Y] := DegOfFromVar. Trans [Y];
    TransMatrixVar. Trans [Z] := DegOfFromVar. Trans [Z]

  END;


  PROCEDURE CalcPosDegOfFromFromTransMatrix (
  VAR DegOfFromVar: DegOfFrom;
  TransMatrixVar: TransMatrix);

    FUNCTION ArcTan2 (
    XValue,
    YValue: Real):
    Real;

    BEGIN

      IF XValue > 0.0
      THEN
      ArcTan2 := ArcTan (YValue / XValue)
      ELSE
      IF XValue < 0.0
      THEN
      ArcTan2 := ArcTan (YValue / XValue) + Pi
      ELSE
      IF YValue > 0.0
      THEN
      ArcTan2 := Pi / 2.0
      ELSE
      ArcTan2 := - Pi / 2.0

    END;

  BEGIN
```

```
    DegOfFromVar. Rotat [Yaw] := ArcTan2 (
    TransMatrixVar. Rotat [X] [X], TransMatrixVar. Rotat [X] [Y]);

    DegOfFromVar. Rotat [Pch] := ArcTan2 (
    SqRt (
    Sqr (TransMatrixVar. Rotat [Y] [Z]) + Sqr (TransMatrixVar. Rotat [Z] [Z])),
    - TransMatrixVar. Rotat [X] [Z]);

    DegOfFromVar. Rotat [R11] := ArcTan2 (
    TransMatrixVar. Rotat [Z] [Z], TransMatrixVar. Rotat [Y] [Z]);

    DegOfFromVar. Trans [X] := TransMatrixVar. Trans [X];
    DegOfFromVar. Trans [Y] := TransMatrixVar. Trans [Y];
    DegOfFromVar. Trans [Z] := TransMatrixVar. Trans [Z]

  END;


BEGIN

  DisableInterrupts;

  Brd2InOperationPtr. SegAddr := - 16386;
  Brd2InOperationPtr. OffAddr :=       0;

  Brd4InOperationPtr. SegAddr := - 8194;
  Brd4InOperationPtr. OffAddr :=      0;

  Brd1InOperationPtr. SegAddr := - 8194;
  Brd1InOperationPtr. OffAddr :=      1;

  Bros12CommIdleBufferBusyFromBrd2Ptr. SegAddr := - 16387;
  Bros12CommIdleBufferBusyFromBrd2Ptr. OffAddr :=       1;
  Bros12CommIdleBufferBusyFromBrd1Ptr. SegAddr := - 16387;
  Bros12CommIdleBufferBusyFromBrd1Ptr. OffAddr :=       0;

  Bros12CommIdleBufferBusyFromBrd1Ptr. AbsAddr^ := False;

  Bros12CommOtptBufferPtrFromBrd2Ptr. SegAddr := - 16388;
  Bros12CommOtptBufferPtrFromBrd2Ptr. OffAddr :=       8;
  Bros12CommIdleBufferPtrFromBrd2Ptr. SegAddr := - 16388;
  Bros12CommIdleBufferPtrFromBrd2Ptr. OffAddr :=       4;
  Bros12CommInptBufferPtrFromBrd2Ptr. SegAddr := - 16388;
  Bros12CommInptBufferPtrFromBrd2Ptr. OffAddr :=       0;
  Bros12CommOtptBufferPtrFromBrd1Ptr. SegAddr := - 16389;
  Bros12CommOtptBufferPtrFromBrd1Prr. OffAddr :=       8;
  Bros12CommIdleBufferPtrFromBrd1Ptr. SegAddr := - 16389;
  Bros12CommIdleBufferPtrFromBrd1Ptr. OffAddr :=       4;
  Bros12CommInptBufferPtrFromBrd1Ptr. SegAddr := - 16389;
  Bros12CommInptBufferPtrFromBrd1Ptr. OffAddr :=       0;

  Bros12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^. SegAddr :=    8185;
  Bros12CommOtptBufferPtrFromBrd2Ptr. AbsAddr^. OffAddr :=       0;
  Bros12CommIdleBufferPtrFromBrd2Ptr. AbsAddr^. SegAddr :=    8183;
  Bros12CommIdleBufferPtrFromBrd2Ptr. AbsAddr^. OffAddr :=       0;
  Bros12CommInptBufferPtrFromBrd2Ptr. AbsAddr^. SegAddr :=    8181;
  Bros12CommInptBufferPtrFromBrd2Ptr. AbsAddr^. OffAddr :=       0;
  Bros12CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. SegAddr := - 16391;
  Bros12CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. OffAddr :=       0;
  Bros12CommIdleBufferPtrFromBrd1Ptr. AbsAddr^. SegAddr := - 16393;
```

```
Brds12CommIdleBufferPtrFromBrd1Ptr. AbsAddr^. OffAddr :=         0;
Brds12CommInptBufferPtrFromBrd1Ptr. AbsAddr^. SegAddr := - 16395;
Brds12CommInptBufferPtrFromBrd1Ptr. AbsAddr^. OffAddr :=         0;

Brds12CommIdleBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. NewData := False;

Brds14CommIdleBufferBusyFromBrd4Ptr. SegAddr := -8195;
Brds14CommIdleBufferBusyFromBrd4Ptr. OffAddr :=     1;
Brds14CommIdleBufferBusyFromBrd1Ptr. SegAddr := -8195;
Brds14CommIdleBufferBusyFromBrd1Ptr. OffAddr :=     0;

Brds14CommIdleBufferBusyFromBrd1Ptr. AbsAddr^ := False;

Brds14CommOtptBufferPtrFromBrd4Ptr. SegAddr := -8196;
Brds14CommOtptBufferPtrFromBrd4Ptr. OffAddr :=     6;
Brds14CommIdleBufferPtrFromBrd4Ptr. SegAddr := -8196;
Brds14CommIdleBufferPtrFromBrd4Ptr. OffAddr :=     4;
Brds14CommInptBufferPtrFromBrd4Ptr. SegAddr := -8196;
Brds14CommInptBufferPtrFromBrd4Ptr. OffAddr :=     0;
Brds14CommOtptBufferPtrFromBrd1Ptr. SegAddr := -8197;
Brds14CommOtptBufferPtrFromBrd1Ptr. OffAddr :=     8;
Brds14CommIdleBufferPtrFromBrd1Ptr. SegAddr := -8197;
Brds14CommIdleBufferPtrFromBrd1Ptr. OffAddr :=     4;
Brds14CommInptBufferPtrFromBrd1Ptr. SegAddr := -8197;
Brds14CommInptBufferPtrFromBrd1Ptr. OffAddr :=     0;

Brds14CommOtptBufferPtrFromBrd4Ptr. AbsAddr^. SegAddr :=  8147;
Brds14CommOtptBufferPtrFromBrd4Ptr. AbsAddr^. OffAddr :=     0;
Brds14CommIdleBufferPtrFromBrd4Ptr. AbsAddr^. SegAddr :=  8107;
Brds14CommIdleBufferPtrFromBrd4Ptr. AbsAddr^. OffAddr :=     0;
Brds14CommInptBufferPtrFromBrd4Ptr. AbsAddr^. SegAddr :=  8067;
Brds14CommInptBufferPtrFromBrd4Ptr. AbsAddr^. OffAddr :=     0;
Brds14CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. SegAddr := -8237;
Brds14CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. OffAddr :=     0;
Brds14CommIdleBufferPtrFromBrd1Ptr. AbsAddr^. SegAddr := -8277;
Brds14CommIdleBufferPtrFromBrd1Ptr. AbsAddr^. OffAddr :=     0;
Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^. SegAddr := -8317;
Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^. OffAddr :=     0;

Brds14CommIdleBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. NewData := False;

Brds41CommIdleBufferBusyFromBrd4Ptr. SegAddr := -8318;
Brds41CommIdleBufferBusyFromBrd4Ptr. OffAddr :=     1;
Brds41CommIdleBufferBusyFromBrd1Ptr. SegAddr := -8318;
Brds41CommIdleBufferBusyFromBrd1Ptr. OffAddr :=     0;

Brds41CommIdleBufferBusyFromBrd1Ptr. AbsAddr^ := False;

Brds41CommOtptBufferPtrFromBrd4Ptr. SegAddr := -8319;
Brds41CommOtptBufferPtrFromBrd4Ptr. OffAddr :=     6;
Brds41CommIdleBufferPtrFromBrd4Ptr. SegAddr := -8319;
Brds41CommIdleBufferPtrFromBrd4Ptr. OffAddr :=     4;
Brds41CommInptBufferPtrFromBrd4Ptr. SegAddr := -8319;
Brds41CommInptBufferPtrFromBrd4Ptr. OffAddr :=     0;
Brds41CommOtptBufferPtrFromBrd1Ptr. SegAddr := -8320;
Brds41CommOtptBufferPtrFromBrd1Ptr. OffAddr :=     8;
Brds41CommIdleBufferPtrFromBrd1Ptr. SegAddr := -8320;
Brds41CommIdleBufferPtrFromBrd1Ptr. OffAddr :=     4;
Brds41CommInptBufferPtrFromBrd1Ptr. SegAddr := -8320;
Brds41CommInptBufferPtrFromBrd1Ptr. OffAddr :=     0;
```

```
ScnnrPosInVenclCrd [VenclX] := ScnnrXOfst;
ScnnrPosInVenclCrd [VenclY] := ScnnrYOfst;
ScnnrPosInVenclCrd [VenclZ] := ScnnrZOfst;

OutByt (0CEH, Chr (0BCH));
OutByt (0CEH, Chr (005H));

writeLn ('The communication port is ready.');

InptSync := False;
OtptSync := False;

OtptSyncChar := Chr (6);

REPEAT

  IF NOT InptSync
  THEN
  BEGIN

    InByt (0CCH, PortCByte. Chrctr);
    IF (PortCByte. BitSet * [Bit5]) <> []
    THEN
    BEGIN

      InptSyncChar2 := InptSyncChar1;
      InptSyncChar1 := InptSyncChar0;
      InByt (0C8H, InptSyncChar0);

      InptSync :=
      (InptSyncChar2 = Chr (2)) AND
      (InptSyncChar1 = Chr (1)) AND
      (InptSyncChar0 = Chr (0))

    END

  END;

  IF NOT OtptSync
  THEN
  BEGIN

    InByt (0CCH, PortCByte. Chrctr);
    IF (PortCByte. BitSet * [Bit0]) <> []
    THEN
    BEGIN

      OtptSyncChar := Chr (Ord (OtptSyncChar) - 1);
      OutByt (0CAH, OtptSyncChar);

      OtptSync :=
      (OtptSyncChar = Chr (0))

    END

  END

UNTIL (InptSync AND OtptSync);

writeLn ('Communications are synchronized.');
```

```
(**
(*   Set up elevation product arrays.
(*)

ElCosProdArrayPtr. SegAddr. IntVal := - 22539;
ElCosProdArrayPtr. OffAddr. LoChar := Chr (000H);
ElSinProdArrayPtr. SegAddr. IntVal := - 22027;
ElSinProdArrayPtr. OffAddr. LoChar := Chr (000H);

FOR ArrayIndex2 := 0 TO 31 DO
BEGIN

  ArrayIndex2TwoChar. IntVal := ArrayIndex2;

  ElCosProdArrayPtr. OffAddr. HiChar := ArrayIndex2TwoChar. LoChar;
  ElSinProdArrayPtr. OffAddr. HiChar := ArrayIndex2TwoChar. LoChar;

  FOR ArrayIndex1 := -RealIInteger TO RealIInteger DO
  BEGIN

    ElCosProdArrayPtr. AbsAddr^ [ArrayIndex1] := Trunc (
    ArrayIndex1 * Cos ((-75.0 + ArrayIndex2 * 60.0 / 31.0) * Pi / 180.0));

    ElSinProdArrayPtr. AbsAddr^ [ArrayIndex1] := Trunc (
    ArrayIndex1 * Sin ((-75.0 + ArrayIndex2 * 60.0 / 31.0) * Pi / 180.0))

  END

END;

(**
(*  Set up azimuth product arrays.
(*)

AzCosProdArrayPtr. SegAddr. IntVal := - 21515;
AzCosProdArrayPtr. OffAddr. LoChar := Chr (000H);
AzSinProdArrayPtr. SegAddr. IntVal := - 21003;
AzSinProdArrayPtr. OffAddr. LoChar := Chr (000H);

FOR ArrayIndex2 := 0 TO  31 DO
BEGIN

  ArrayIndex2TwoChar. IntVal := ArrayIndex2;

  AzCosProdArrayPtr. OffAddr. HiChar := ArrayIndex2TwoChar. LoChar;
  AzSinProdArrayPtr. OffAddr. HiChar := ArrayIndex2TwoChar. LoChar;

  FOR ArrayIndex1 := -RealIInteger TO RealIInteger DO
  BEGIN

    AzCosProdArrayPtr. AbsAddr^ [ArrayIndex1] := Trunc (
    ArrayIndex1 * Cos ((40.0 - ArrayIndex2 * 80.0 / 31.0) * Pi / 180.0));

    AzSinProdArrayPtr. AbsAddr^ [ArrayIndex1] := Trunc (
    ArrayIndex1 * Sin ((40.0 - ArrayIndex2 * 80.0 / 31.0) * Pi / 180.0))

  END

END;

(**
(*  Set up scan point displacement array.
(*)

ScanPtDispArrayPtr. SegAddr. IntVal := - 20491;
ScanPtDispArrayPtr. OffAddr. LoChar := Chr (000h);

FOR ArrayIndex2 := 1 TO 255 DO
BEGIN

  ArrayIndex2TwoChar. IntVal := ArrayIndex2;

  ScanPtDispArrayPtr. OffAddr. HiChar := ArrayIndex2TwoChar. LoChar;

  FOR ArrayIndex1 := -RealIInteger TO RealIInteger DO
  ScanPtDispArrayPtr. AbsAddr^ [ArrayIndex1] :=
  ArrayIndex1 * ArrayIndex2 DIV RealIInteger

END;

WriteLn ('Board 1 is in operation.');

Brd1InOperationPtr. AbsAddr^ := False;

FOR LegIdx := FtLt TO RrRt DO
Brds12CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
VehclLegsCmnds [LegIdx]. SptStt := Support;

WHILE True DO
BEGIN

  FOR CrdIndex1 := X TO Z DO
  FOR CrdIndex2 := X TO Z DO
  InReal (VehclToEarthTransMatrix. Rotat [CrdIndex1] [CrdIndex2]);

  FOR CrdIndex := X TO Z DO
  InReal (VehclToEarthTransMatrix. Trans [CrdIndex]);

  FOR VehclCrdIndex := VehclX TO VehclZ DO
  InReal (VehclLinVelInVehclCrd [VehclCrdIndex]);

  FOR VehclCrdIndex := VehclX TO VehclZ DO
  InReal (VehclAngVelInVehclCrd [VehclCrdIndex]);

  InReal (CurrntTime);

  FOR CrdIndex1 := X TO Z DO
  FOR CrdIndex2 := X TO Z DO
  Brds12CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
  ScnnrToEarthTransMatrix. Rotat [CrdIndex1] [CrdIndex2] := Round (
  VehclToEarthTransMatrix. Rotat [CrdIndex1] [CrdIndex2] * RealIInteger);

  TrnsfrmPtToEarthCrdFrVehclCrd (ScnnrPosInEarthCrd,
  VehclToEarthTransMatrix, ScnnrPosInVehclCrd);

  Brds12CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
  ScnnrToEarthTransMatrix. Trans [X] := Round (
  ScnnrPosInEarthCrd [EarthX] * 8.0);
  Brds12CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
  ScnnrToEarthTransMatrix. Trans [Y] := Round (
  ScnnrPosInEarthCrd [EarthY] * 8.0);
  Brds12CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
```

```
SchnrToEarthTransMatrix. Trans [Z] := Round (
SchnrPosInEarthCrd [EarthZ] * 8.0);

REPEAT

  Brds12CommIdleBufferBusyFromBrd1Ptr. AbsAddr^ := True;

  IF Brds12CommIdleBufferBusyFromBrd2Ptr. AbsAddr^
  THEN
  Brds12CommIdleBufferBusyFromBrd1Ptr. AbsAddr^ := False

UNTIL Brds12CommIdleBufferBusyFromBrd1Ptr. AbsAddr^;

Brds12CommTempBufferPtr :=
Brds12CommInptBufferPtrFromBrd1Ptr. AbsAddr^;

Brds12CommInptBufferPtrFromBrd1Ptr. AbsAddr^ :=
Brds12CommIdleBufferPtrFromBrd1Ptr. AbsAddr^;

Brds12CommIdleBufferPtrFromBrd1Ptr. AbsAddr^ :=
Brds12CommTempBufferPtr;

Brds12CommTempBufferPtr :=
Brds12CommInptBufferPtrFromBrd2Ptr. AbsAddr^;

Brds12CommInptBufferPtrFromBrd2Ptr. AbsAddr^ :=
Brds12CommIdleBufferPtrFromBrd2Ptr. AbsAddr^;

Brds12CommIdleBufferPtrFromBrd2Ptr. AbsAddr^ :=
Brds12CommTempBufferPtr;

Brds12CommIdleBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. NewData := True;

Brds12CommIdleBufferBusyFromBrd1Ptr. AbsAddr^ := False;

Brd2InOperationPtr. AbsAddr^ := True;

InChar (OperatingMode);

InChar (SyncChar);

IF SyncChar <> Chr (0AAH)
THEN
WriteLn ('*** Communication synchronization error ***');

Status := 'A';

OutChar (Status);

SyncChar := Chr (0AAH);

OutChar (SyncChar);

IF (OperatingMode = '1')
THEN
BEGIN

  Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. CurrntTime :=
  CurrntTime;

  Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
```

```
VehclToEarthTransMatrix := VehclToEarthTransMatrix;

Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
VehclLinVelInVehclCrd := VehclLinVelInVehclCrd;

Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
VehclAngVelInVehclCrd := VehclAngVelInVehclCrd;

FOR LegIndex := FtLt TO RrRt DO
BEGIN

  FOR VehclCrdIndex := VehclX TO VehclZ DO
  InReal (LegPosInVehclCrd [VehclCrdIndex]);

  TrnsfrmPtToEarthCrdFrVehclCrd (
  Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
  VehclLegsStts [LegIndex]. PosInEarthCrd ,
  VehclToEarthTransMatrix, LegPosInVehclCrd);

  InItgr (SptSttItgr);

  IF SptSttItgr = 0
  THEN
  VehclLegsSptStts [LegIndex] := Support
  ELSE
  VehclLegsSptStts [LegIndex] := Trnsfer;

  Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
  VehclLegsStts [LegIndex]. SptStt := VehclLegsSptStts [LegIndex]

END;

InReal (Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
FrwdVelRqst);
InReal (Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
SideVelRqst);
InReal (DummyReal);
InReal (DummyReal);
InReal (DummyReal);
InReal (Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
TurnVelRqst);

REPEAT

  Brds14CommIdleBufferBusyFromBrd1Ptr. AbsAddr^ := True;

  IF Brds14CommIdleBufferBusyFromBrd4Ptr. AbsAddr^
  THEN
  Brds14CommIdleBufferBusyFromBrd1Ptr. AbsAddr^ := False

UNTIL Brds14CommIdleBufferBusyFromBrd1Ptr. AbsAddr^;

Brds14CommTempBufferPtr :=
Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^;

Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^ :=
Brds14CommIdleBufferPtrFromBrd1Ptr. AbsAddr^;

Brds14CommIdleBufferPtrFromBrd1Ptr. AbsAddr^ :=
Brds14CommTempBufferPtr;
```

```
Brds14CommTempBufferPtr :=
Brds14CommInptBufferPtrFromBrd4Ptr. AbsAddr^;

Brds14CommInptBufferPtrFromBrd4Ptr. AbsAddr^ :=
Brds14CommIdleBufferPtrFromBrd4Ptr. AbsAddr^;

Brds14CommIdleBufferPtrFromBrd4Ptr. AbsAddr^ :=
Brds14CommTempBufferPtr;

Brds14CommIdleBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. NewData := True;

Brds14CommIdleBufferBusyFromBrd1Ptr. AbsAddr^ := False;

Brd4InOperationPtr. AbsAddr^ := True;

Brd1InOperationPtr. AbsAddr^ := False;

WriteLn ('Board 4 is in operation.');

REPEAT
UNTIL Brd1InOperationPtr. AbsAddr^;

Brds41CommIdleBufferBusyFromBrd1Ptr. AbsAddr^ := True;

REPEAT
UNTIL NOT Brds41CommIdleBufferBusyFromBrd4Ptr. AbsAddr^;

IF Brds41CommIdleBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. NewData
THEN
BEGIN

  Brds41CommTempBufferPtr :=
  Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^;

  Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^ :=
  Brds41CommIdleBufferPtrFromBrd1Ptr. AbsAddr^;

  Brds41CommIdleBufferPtrFromBrd1Ptr. AbsAddr^ :=
  Brds41CommTempBufferPtr;

  Brds41CommTempBufferPtr :=
  Brds41CommOtptBufferPtrFromBrd4Ptr. AbsAddr^;

  Brds41CommOtptBufferPtrFromBrd4Ptr. AbsAddr^ :=
  Brds41CommIdleBufferPtrFromBrd4Ptr. AbsAddr^;

  Brds41CommIdleBufferPtrFromBrd4Ptr. AbsAddr^ :=
  Brds41CommTempBufferPtr;

  Brds41CommIdleBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. NewData :=
  False

END;

Brds41CommIdleBufferBusyFromBrd1Ptr. AbsAddr^ := False;

BdySttIdxFound := False;
BdySttIdx := 0;

WHILE (NOT BdySttIdxFound) AND
(BdySttIdx <
Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. VehclBdyTraj.
MaxBdySttIdx) DO
IF
(CurrntTime >=
Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. VehclBdyTraj.
VehclBdyStts [BdySttIdx + 1]. Time)
THEN
BdySttIdx := BdySttIdx + 1
ELSE
BdySttIdxFound := True;

CalcPosDegOfFrdmFromTransMatrix (
LastVehclPosDegOfFrdmWRTEarthCrd,
Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. VehclBdyTraj.
VehclBdyStts [BdySttIdx]. VehclToEarthTransMatrix);

IF
BdySttIdx =
Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. VehclBdyTraj.
MaxBdySttIdx
THEN
BEGIN

  CrntVehclPosDegOfFrdmWRTEarthCrd := LastVehclPosDegOfFrdmWRTEarthCrd;

  CrntVehclVelDegOfFromWRTEarthCrd. Rotat [Yaw] := 0.0;
  CrntVehclVelDegOfFromWRTEarthCrd. Rotat [Pch] := 0.0;
  CrntVehclVelDegOfFromWRTEarthCrd. Rotat [Rll] := 0.0;
  CrntVehclVelDegOfFromWRTEarthCrd. Trans [X]   := 0.0;
  CrntVehclVelDegOfFromWRTEarthCrd. Trans [Y]   := 0.0;
  CrntVehclVelDegOfFromWRTEarthCrd. Trans [Z]   := 0.0

END
ELSE
BEGIN

  CalcPosDegOfFrdmFromTransMatrix (
  NextVehclPosDegOfFrdmWRTEarthCrd,
  Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. VehclBdyTraj.
  VehclBdyStts [BdySttIdx + 1]. VehclToEarthTransMatrix);

  IntrpltnTimeRatio :=
  (CurrntTime -
  Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. VehclBdyTraj.
  VehclBdyStts [BdySttIdx]. Time) /
  (Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. VehclBdyTraj.
  VehclBdyStts [BdySttIdx + 1]. Time -
  Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. VehclBdyTraj.
  VehclBdyStts [BdySttIdx]. Time);

  CrntVehclPosDegOfFrdmWRTEarthCrd. Rotat [Yaw] :=
  LastVehclPosDegOfFromWRTEarthCrd. Rotat [Yaw] + (
  NextVehclPosDegOfFromWRTEarthCrd. Rotat [Yaw] -
  LastVehclPosDegOfFromWRTEarthCrd. Rotat [Yaw]) * IntrpltnTimeRatio;
  CrntVehclPosDegOfFromWRTEarthCrd. Rotat [Pch] :=
  LastVehclPosDegOfFromWRTEarthCrd. Rotat [Pch] + (
  NextVehclPosDegOfFromWRTEarthCrd. Rotat [Pch] -
  LastVehclPosDegOfFromWRTEarthCrd. Rotat [Pch]) * IntrpltnTimeRatio;
  CrntVehclPosDegOfFrdmWRTEarthCrd. Rotat [Rll] :=
  LastVehclPosDegOfFromWRTEarthCrd. Rotat [Rll] + (
  NextVehclPosDegOfFromWRTEarthCrd. Rotat [Rll] -
```

```
      LastVehclPosDegOfFromWRTEarthCrd. Rotat [Rll]) * IntrpltnTimeRatio;
      CrntVehclPosDegOfFromWRTEarthCrd. Trans [X] :=
      LastVehclPosDegOfFromWRTEarthCrd. Trans [X] + (
      NextVehclPosDegOfFromWRTEarthCrd. Trans [X] -
      LastVehclPosDegOfFromWRTEarthCrd. Trans [X]) * IntrpltnTimeRatio;
      CrntVehclPosDegOfFromWRTEarthCrd. Trans [Y] :=
      LastVehclPosDegOfFromWRTEarthCrd. Trans [Y] + (
      NextVehclPosDegOfFromWRTEarthCrd. Trans [Y] -
      LastVehclPosDegOfFromWRTEarthCrd. Trans [Y]) * IntrpltnTimeRatio;
      CrntVehclPosDegOfFromWRTEarthCrd. Trans [Z] :=
      LastVehclPosDegOfFromWRTEarthCrd. Trans [Z] + (
      NextVehclPosDegOfFromWRTEarthCrd. Trans [Z] -
      LastVehclPosDegOfFromWRTEarthCrd. Trans [Z]) * IntrpltnTimeRatio;

      VelCalcTimeIntrvl :=
      Brds4iCommOtotBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^. VehclBdyTraj.
      VehclBdyStts [BdySttIdx + 1]. Time -
      CurrntTime;

      CrntVehclVelDegOfFromWRTEarthCrd. Rotat [Yaw] := (
      NextVehclPosDegOfFromWRTEarthCrd. Rotat [Yaw] -
      CrntVehclPosDegOfFromWRTEarthCrd. Rotat [Yaw]) / VelCalcTimeIntrvl;
      CrntVehclVelDegOfFromWRTEarthCrd. Rotat [Pch] := (
      NextVehclPosDegOfFromWRTEarthCrd. Rotat [Pch] -
      CrntVehclPosDegOfFromWRTEarthCrd. Rotat [Pch]) / VelCalcTimeIntrvl;
      CrntVehclVelDegOfFromWRTEarthCrd. Rotat [Rll] := (
      NextVehclPosDegOfFromWRTEarthCrd. Rotat [Rll] -
      CrntVehclPosDegOfFromWRTEarthCrd. Rotat [Rll]) / VelCalcTimeIntrvl;
      CrntVehclVelDegOfFromWRTEarthCrd. Trans [X] := (
      NextVehclPosDegOfFromWRTEarthCrd. Trans [X] -
      CrntVehclPosDegOfFromWRTEarthCrd. Trans [X]) / VelCalcTimeIntrvl;
      CrntVehclVelDegOfFromWRTEarthCrd. Trans [Y] := (
      NextVehclPosDegOfFromWRTEarthCrd. Trans [Y] -
      CrntVehclPosDegOfFromWRTEarthCrd. Trans [Y]) / VelCalcTimeIntrvl;
      CrntVehclVelDegOfFromWRTEarthCrd. Trans [Z] := (
      NextVehclPosDegOfFromWRTEarthCrd. Trans [Z] -
      CrntVehclPosDegOfFromWRTEarthCrd. Trans [Z]) / VelCalcTimeIntrvl

   End;

   CalcTransMatrixFromPosDegOfFrm (
   CrntVehclToEarthTransMatrix,
   CrntVehclPosDegOfFromWRTEarthCrd);

   VehclLinVelInEarthCrdCmnd [EarthX] :=
   CrntVehclVelDegOfFromWRTEarthCrd. Trans [X];
   VehclLinVelInEarthCrdCmnd [EarthY] :=
   CrntVehclVelDegOfFromWRTEarthCrd. Trans [Y];
   VehclLinVelInEarthCrdCmnd [EarthZ] :=
   CrntVehclVelDegOfFromWRTEarthCrd. Trans [Z];

   TrnsfrmVcToVehclCrdFrEarthCrd (VehclLinVelInVehclCrdCmnd,
   CrntVehclToEarthTransMatrix, VehclLinVelInEarthCrdCmnd);

   VehclAngVelInVehclCrdCmnd [VehclX] :=
   CrntVehclVelDegOfFromWRTEarthCrd. Rotat [Rll];
   VehclAngVelInVehclCrdCmnd [VehclY] :=
   CrntVehclVelDegOfFromWRTEarthCrd. Rotat [Pch];
   VehclAngVelInVehclCrdCmnd [VehclZ] :=
   CrntVehclVelDegOfFromWRTEarthCrd. Rotat [Yaw];
```

```
FOR LegIdx := FtLt TO RrRt DO
BEGIN

   LegTrajIdxFound := False;
   LegsTrajIdx [LegIdx] := 0;

   WHILE (NOT LegTrajIdxFound) AND
   (LegsTrajIdx [LegIdx] <
   Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
   VenclLegsTrajs [LegIdx]. MaxLegTrajIdx) DO
   IF
   (CurrntTime >=
   Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
   VenclLegsTrajs [LegIdx]. VenclLegTrajs [LegsTrajIdx [LegIdx]]. CtTime)
   THEN
   LegsTrajIdx [LegIdx] := LegsTrajIdx [LegIdx] + 1
   ELSE
   LegTrajIdxFound := True;

   IF LegsTrajIdx [LegIdx] >
   Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
   VehclLegsTrajs [LegIdx]. MaxLegTrajIdx
   THEN
   LegsTrajIdx [LegIdx] := NoTraj
   ELSE
   IF NOT
   ((CurrntTime >=
   Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
   VehclLegsTrajs [LegIdx]. VenclLegTrajs [LegsTrajIdx [LegIdx]].
   CtTime)
   AND
   (CurrntTime <
   Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
   VenclLegsTrajs [LegIdx]. VenclLegTrajs [LegsTrajIdx [LegIdx]].
   CtTime))
   THEN
   LegsTrajIdx [LegIdx] := NoTraj

END;

FOR CrdIndex1 := X TO Z DO
FOR CrdIndex2 := X TO Z DO
WITH CrntVenclToEarthTransMatrix DO
OutReal (Rotat [CrdIndex1] [CrdIndex2]);

FOR CrdIndex := X TO Z DO
WITH CrntVehclToEarthTransMatrix DO
OutReal (Trans [CrdIndex]);

FOR VenclCrdIndex := VenclX TO VenclZ DO
OutReal (VenclLinVelInVenclCrdCmnd [VenclCrdIndex]);

FOR VehclCrdIndex := VenclX TO VehclZ DO
OutReal (VehclAngVelInVenclCrdCmnd [VenclCrdIndex]);

FOR LegIdx := FtLt TO RrRt DO
BEGIN

   IF
   (LegsTrajIdx [LegIdx] = NoTraj) OR
   (VenclLegsSptStts [LegIdx] = Trnsfer)
```

```
          THEN
          BEGIN

            OutItgr (Ord (NOT True));

            Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
            VehclLegsCmnds [LegIdx]. SptStt := Support

          END
          ELSE
          BEGIN

            OutItgr (Ord (NOT False));

            Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
            VehclLegsCmnds [LegIdx]. SptStt := Trnsfer

          END;

          IF NOT (LegsTrajIdx [LegIdx] = NoTraj)
          THEN
          Brds14CommInptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
          VehclLegsCmnds [LegIdx]. VehclLegCmndTraj :=
          Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
          VehclLegsTrajs [LegIdx]. VehclLegTrajs [LegsTrajIdx [LegIdx]];

          WITH
          Brds41CommOtptBufferPtrFromBrd1Ptr. AbsAddr^. AbsAddr^.
          VehclLegsTrajs [LegIdx]. VehclLegTrajs [LegsTrajIdx [LegIdx]] DO
          BEGIN

            OutReal (CttTime);

(*          OutReal (LftTime);
*)          OutReal (LftHgt);

(*          OutReal (PlcTime);
*)          FOR VehclCrdIndex := VehclX TO VehclZ DO
            OutReal (PlcPosInVehclCrd [VehclCrdIndex]);

(*          OutReal (CttTime);
*)          OutReal (CttHgtMin);
            OutReal (CttHgtMax);
(*          FOR EarthCrdIndex := EarthX TO EarthZ DO
            OutReal (NxtFhoInEarthCrd [EarthCrdIndex])
*)
          END
        END
      END
      ELSE
      BEGIN

        Brd4InOperationPtr. AbsAddr^ := False

      END

    END

END.
```

APPENDIX H

FORCES ON SPOOL

FORCES ON SPOOL

Moving Forces

Spring Force

F start $= \kappa\Delta x = 88$ lb/in (0.258 in) = 22 lbf

F finish $= \kappa\Delta x = 88$ lb/in (0.883 in) = 78 lbf

Rod Area Differential

F = PA $\qquad A = \frac{\pi d^2}{4} = \frac{\pi(0.25 \text{ in})^2}{4} = 0.05 \text{ in}^2$

F max = 4000 psi (0.05 in^2) = 220 lbf

F min = 100 psi (0.05 in^2) = 6 lbf

Opposing Forces

Seal Force (from two components)

F seal = F compression + F pressure = moving force on seal

F compression = 1.5 lb/in (0.25 in) π = 1.2 lbf

F pressure (4500) = 70 lb/in $\left(\frac{4500 \text{ psi}}{3000 \text{ psi}}\right)(0.375 \text{ in})^2(.785) = 11.6$ lbf

(100) = 0.25 lbf

Static Friction is approximately 3 times moving friction

F seal moving = 1.5 lbf @ 100 psi

= 12.8 lbf @ 4500 psi

F seal static = 4.5 lbf @ 100 psi

= 38.4 lbf @ 4500 psi

Hydraulic Drag Forces

F hyd. Drag = ΔPA $\qquad A = (1.0 \text{ in})^2 (.785) - .785 \text{ in}^2$

ΔP is a function of flow and hole size. For the spools there are six 11/64-inch holes used for moving fluid. The flow is the volume of fluid moved in the valve shift time.

$$Q = (0.785 \text{ in}^2)(0.625 \text{ in})/0.01 \text{ sec} = 49 \text{ in}^3/\text{sec}$$
$$= 12.7 \text{ GPM}$$

We can find the pressure drop by looking at flow through orifice charts

$$\Delta P = 13 \text{ psi}$$

$$F \text{ hyd. Drag} = (13 \text{ psi})(0.785) = 10.2 \text{ lbf}$$

Since this force is proportional to Q^2 then this would be the worst case because we assumed a 10 msec shift time instead of 50 msec, the actual force would be much lower.

Battelle
Columbus Laboratories
505 King Avenue
Columbus, Ohio 43201
Telephone (614) 424-6424
Telex 24-5454

March 13, 1984

U.S. Army Tank-Automotive Command
Attention: DRSTA-ZSS
Warren, Michigan 48090

Dear Sir/Madam:

Reference: Contract Number DAAE07-83-C-R040
DARPA order number 4670

According to the contractual requirements copies of the Final Report on the referenced contract were distributed as indicated on the attached distribution list. The corrections were made according to telephone directions from TACOM.

Battelle Columbus Laboratories has enjoyed working on this program and look forward to participating on future programs. Project questions should be directed to Dr. Richard K. Thatcher, who can be contacted at the above address, or by telephone at (614) 424-7750.

Sincerely,

Richard K. Thatcher

Richard K. Thatcher
Associate Section Manager
Digital Systems and Technology Section

RKT:jwm

cc: U.S. Army Tank-Automotive Command
Attention: DRSTA-I, Buyer Code:RRRD
P & P Directorate
Warren, Michigan 48090

Defense Logistics Agency
DCASMA, Dayton
Attention: Mr. Jesse L. Richey, DCRO-GDCA-J3
c/o Defense Electronics Supply Center
Dayton, Ohio 45444

DISTRIBUTION LIST

Address	Copies
U.S. Army Tank-Automotive Command Attention: DRSTA-ZSS Warren, Michigan 48090 M/F: DAAE07-83-C-R040	11
U.S. Army Tank-Automotive Command Attention: DRSTA-FSL Warren, Michigan 48090	2
Director Defense Advanced Research Project Agency Attention: TIO/Admin 1400 Wilson Blvd. Arlington, Virginia 22209	3
Director Defense Advanced Research Project AGency Attention: Dr. Clinton Kelly 1400 Wilson Blvd. Arliongton, Virginia 22209	15
Defense Documentation Center Cameron Station Alexandria, Virginia 22314	11/1
TACTEC Battelle Memorial Institute 505 King Avenue Columbus, Ohio 43201	1